222

Tips, Tricks & Tools

für den C64

Nikolaus M.Heusler

In Zusammenarbeit mit dem 64'er-Magazin

IPV

****

Welcome to International Project 64!

The goal of International Project 64 is to preserve non-English Commodore 64 related documents in electronic text format that might otherwise cease to exist with the rapid advancement of computer technology and declining interest in 8-bit computers on the part of the general population. If you would like to help by converting C64 related hardcopy documents to electronic texts please contact the manager of International Project 64, Peter Karlsson, at pk@abc.se.

Extensive efforts were made to preserve the contents of the original document. However, certain portions, such as diagrams, program listings, and indexes may have been either altered or sacrificed due to the limitations of plain vanilla text. Diagrams may have been eliminated where ASCII-art was not feasible. Program listings may be missing display codes where substitutions were not possible. Tables of contents and indexes may have been changed from page number references to section number references. Please accept our apologies for these limitations, alterations, and possible omissions.

Document names are limited to the 8.3 file convention of DOS. The first characters of the file name are an abbreviation of the original document name and the language of the etext. The version number of the etext follows next. After that a letter may appear to indicate the particular source of the document. Finally, the document is given a
.TXT extension.

The author(s) of the original document and members of International Project 64 make no representations about the accuracy or suitability of this material for any purpose. This etext is provided "as-is". Please refer to the warantee of the original document, if any, that may included in this etext. No other warantees, express or implied, are made to you as to the etext or any medium it may be on. Neither the author(s) nor the members of International Project 64 will assume liability for damages either from the direct or indirect use of this etext or from the distribution of or modification to this etext. Therefore if you read this document or use the information herein you do so at your own risk.

****

The International Project 64 etext of the book "222 Tips, Tricks und Tools für den C 64", written and donated for IP64 distribution by Nikolaus Heusler <nikolaus.heusler@lrz.tu-muenchen.de>.

222_DE1.TXT, June 1997, etext #8.

Note from the etexter:

This is german text.

This document is the text of a book written by Nikolaus M. Heusler. The book was published in 1992 and contains information about the C 64. No part of this text may be copied, stored or published in any way without prior written permission by the author. It is not allowed to store copies of this text or of parts of it on computer-discettes or on computers which can be accessed in the public, for example on the internet.

Das Werk einschliesslich aller seiner Teile ist urheberrechtlich geschuetzt. Jede Verwertung ausserhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne vorherige schriftliche Zustimmung des Verfassers unzulaessig und strafbar. Dies gilt insbesondere fuer Vervielfaeltigungen, Uebersetzungen und die Verarbeitung in elektronischen Systemen.

Einwilligungen zur Weiterverarbeitung wird der Verfasser im Einzelfall auf schriftliche Anfrage gern geben, falls keine kommerziellen Interessen mit der Verarbeitung verfolgt werden.

Produkt- und Herstellerangaben in dem Buch werden ohne Ruecksicht auf eventuellen Patentschutz veroeffentlicht. Warennamen werden ohne Gewaehrleistung der freien Verwendbarkeit benutzt. Die Texte wurden mit groesster Sorgfalt zusammengestellt. Trotzdem sind Fehler nicht vollstaendig auszuschliessen. Fuer fehlerhafte Angaben und deren Folgen kann der Autor weder juristische Verantwortung noch irgendeine Haftung uebernehmen.

Zu dem Buch gehoeren zwei Disketten im 1541-Format mit den Listings aus dem Buch.

Das Buch selbst ist leider nicht mehr lieferbar.

Verfasser / Author: Nikolaus Heusler
Zwengauerweg 18
D-81479 Muenchen
Germany

Bei schriftlichen Anfragen bitte Rueckporto beilegen, ich antworte gern.

This version was published in the internet on the project "IP 64" in June 1997.

****

222 Tips, Tricks und Tools fuer den C 64

INHALTSVERZEICHNIS

Diskettenbriefe - der Message-Maker
Die große Trickkiste - auch für Sie ist etwas dabei!

6. Das große Computerlexikon

Vorwort

Daran gibt es keinen Zweifel: Der C 64 ist Weltrekordhalter beim Verkauf von Homecomputern. Mit ziemlicher Sicherheit wird es auch in der näheren Zukunft keinen Rechner geben, der von der Verbreitung her an die über zehn Millionen C 64 herankommt. Ein wichtiger Grund dafür ist sicherlich die für seinen Preis ungewöhnlich gute technische Ausstattung, was Speicher, Grafikfähigkeit und die Soundmöglichkeiten anbelangt.

Ein großes Manko war jedoch schon immer das Handbuch zum C 64. Der frischgebackene Besitzer wird nur sehr grob in die wichtigsten Grundzüge des eingebauten Basic-Dialekts eingeführt. Die Bedienungsanleitung geht weder auf die Programmierung in Maschinensprache, die viele Anwendungen erst ermöglicht, ein, noch steht etwas über die Programmierung der hochauflösenden Grafik drin. Und das sparsame Basic V2 unterstützt die enorme Leistungsfähigkeit kaum. Will man ernsthaft mit dem neuen Gerät arbeiten und programmieren, ist man auf zusätzliche Literatur angewiesen. Fachzeitschriften wie das 64'er-Magazin bieten regelmäßig interessante Tips und Tricks, mit denen schnell die ersten Erfolgserlebnisse kommen.

Dieses Buch ist eine weitere wichtige und sehr ergiebige Informationsquelle für Anwender, vor allem aber Programmierer. Enthalten sind unzählige kleine, größere und umfangreiche Programme, die Ihren Programmieralltag erheblich erleichtern werden. Nicht nur der Einsteiger findet hier die Anregungen und Tips, die das Handbuch ausläßt. Was halten Sie von einem Tool, das alle Basic-Fehlermeldungen in deutsch näher erläutert? Tastatur-Utilities erlauben die Belegung der Funktionstasten oder das Abrufen von Basicbefehlen auf Tastendruck. Auch für den Fortgeschrittenen ist viel dabei. Obwohl praktische Hilfen für die Basicprogrammierung den Schwerpunkt darstellen, ist auch einiges aus dem Bereich der Maschinensprache geboten, etwa ein kompletter Kurs über die Dateiprogrammierung in Assembler. Natürlich soll auch der Profi nicht zu kurz kommen, als Beispiele seien hier nur »DOC 64« und der »Tester 1541« genannt, zwei Programme, die den C 64 und das Laufwerk auf Herz und Nieren überprüfen, oder ein interessantes Multitask-Hilfsprogramm, mit dessen Hilfe mehrere Projekte gleichzeitig ablaufen. Zum Thema Floppyprogrammierung gibt es allerlei Datei-Analyseprogramme, zwei Diskmonitoren, die die geheimnisvollen schwarzen Scheiben entzaubern, oder ein Hilfsprogramm zum Retten gelöschter Files.

Das alles geht ohne stundenlanges Abtippen: Auf den beiliegenden Disketten finden Sie alle Programme fertig zum Ausprobieren. Obwohl viele Utilities weitgehend selbsterklärend sind, lassen sich jedoch mit Hilfe der ausführlichen Anleitungen ärgerliche Bedienungsfehler vermeiden.

Mein besonderer Dank gilt den vielen Lesern meiner Beiträge im 64'er-Magazin, die mit konstruktiver Kritik, Verbesserungsvorschlägen und wichtigen Anregungen in Form von Leserbriefen entscheidend dazu beigetragen haben, dieses Buch auf einen sehr breiten Leserkreis auszurichten. Dank gebührt außerdem den Mitarbeitern des IPV-Verlags, vor allem den Herren

Felix Röscheisen und Axel Pretzsch, die mit viel Liebe und Sorgfalt die Texte zusammengestellt und das Buch produziert haben. Seitens des Markt & Technik Verlags bedanke ich mich vor allem bei der Redaktion des 64'er-Magazins für die sehr gute und freundschaftliche Zusammenarbeit.

Ich hoffe, daß dieses Buch dem Einsteiger eine gute Hilfe beim Erarbeiten und zum Verständnis der C 64-Materie und dem Fortgeschrittenen ein brauchbares Nachschlagewerk ist. Gerne nehme ich Kritik und Verbesserungsvorschläge entgegen.

München-Solln, im August 1992

Nikolaus M. Heusler

KAPITEL 1: BASIC-PROGRAMMIERHILFEN

Der Detektiv für Basicprogramme: Anleitung zum Basic-Kontroll System (BKS), Version 5.0

Gleich der erste Beitrag im Buch ist ein Highlight: Ein hervorragendes Hilfsprogramm prüft Ihre Programme. Fehlerhafte Zeilen werden angezeigt und auf Wunsch ausgedruckt. Die zweitraubende und schweißtreibende Fehlersuche in einem Programm wird damit kinderleicht.

Das kurze Maschinenprogramm geht ein im Speicher stehendes Basicprogramm Zeile für Zeile durch und testet dabei, ob typische Fehler enthalten sind, die von vielen Leuten gemacht werden, beispielsweise: Klammer zu viel oder wenig, Syntaxfehler, Formatfehler im Programmtext. Ganz besonders hat es die Testroutine auf Sprungbefehle abgesehen. Hier werden eigentlich alle möglichen Fehler gefunden. Dieses Programm wurde von mir als redaktionellem Mitarbeiter eigentlich entwickelt, um Basic-Leserlistings für das 64'er-Magazin auf ihre Lauffähigkeit zu testen.

Doch das BKS (Basic Kontroll System) ist sicherlich nicht nur für professionelle Programmtester interessant. Auch der »Privatmensch« kann durchaus etwas damit anfangen. Beispielsweise ein Einsteiger, der gerne überprüfen möchte, ob das Basicprogramm, das er gerade geschrieben hat, auch sicher läuft. Auch der Fortgeschrittene hat etwas davon. Nehmen wir einmal an, Sie haben ein längeres (vielleicht sogar fremdes) Basicprogramm vorliegen, das Sie überarbeiten möchten. Sie nehmen hie und da eine Zeile heraus, fügen dort eine neue ein, und vertauschen einige Programmteile. Doch was ist, wenn Sie dabei irgendwo einen GOTO-Befehl übersehen haben, der jetzt auf eine nicht mehr existierende Zeile weist? Das Programm wird plötzlich aussteigen, wie üblich dann, wenn man es am wenigsten braucht.

Wenn Sie bereits Profi sind, kennen Sie sicher folgende Situation: Sie haben ein langes und komplexes Basicprogramm geschrieben, das von Maschinenprogrammen unterstützt wird. Da wird dann eifrig im Speicher herumgepoket, der Programmierer wirft mit SYS-Befehlen um sich, ein Finger hat seinen Stammplatz am RESET-Taster. Es ist bereits 2 Uhr nachts. Hopps ! Da haben Sie in der Eile einen falschen POKE-Befehl eingegeben, der das Basicprogramm angegriffen hat. Oder durch eine ungeschickte Manipulation wurde der Basic-Endezeiger 45/46 so verändert, daß ein Start des Programmes

den totalen Absturz zur Folge hat. In solch einem Fall lädt man einen Monitor und versucht, zu retten, was noch zu retten ist. Doch stimmen alle Zeiger, alle Zeilennummern, alle Sprünge jetzt noch?

In allen diesen Fällen können Sie sich mit dem BKS behelfen. Obwohl das BKS aus Gründen der Geschwindigkeit und des Komforts vollständig in stark optimierter Maschinensprache verfaßt wurde, brauchen Sie zur Anwendung keinerlei Assemblerkenntnisse. Laden Sie die Testroutine mit

```
LOAD "BKS 5.0 (49152)",8,1
```

Danach geben Sie bitte NEW ein, um alle Pointer richtigzustellen. Keine Angst, das Maschinenprogramm wird dabei nicht gelöscht. Es kann jederzeit mit SYS 49152 gestartet werden, allerdings nur, wenn sich ein Basicprogramm im Speicher befindet. Sonst wird eine entsprechende Meldung ausgegeben.

Laden Sie nun also das zu testende Basicprogramm. Der Stand des Basic-Anfangszeigers (43/44) spielt keine Rolle. Starten Sie nun das BKS mit SYS 49152. Es erscheinen nun einige Fragen, die Sie durch Druck auf die Taste <J> für Ja oder <N> für Nein beantworten. Zunächst werden Sie gefragt, ob Spaces bemängelt werden sollen. Manche Programmierer setzen in Basicprogramme viele Leerzeichen, um sie übersichtlicher zu machen, andere verschmähen dies. Sollten Sie zu letzteren gehören, können Sie die Frage mit <J> beantworten, dann wird auch getestet, ob außerhalb von Anführungszeichen und DATAs überflüssige Spaces auftreten. Wenn in einem Programm allerdings zu viele Leerzeichen sind, besteht die Liste, die das BKS erzeugt, praktisch nur noch aus diesem Fehler, und wird unübersichtlich. Schalten Sie den Fehler dann also besser aus.

Die nächste Frage hat eine ähnliche Funktion. Das BKS kann auf Wunsch auch Sprünge (RUN, GOTO, GOSUB, THEN) finden, die auf eine REM-Zeile oder eine Trennzeile, das ist eine Zeile dieser Art:

```
10 :
```

weisen. Das kann problematisch werden, wenn das Programm mit einem »unintelligenten« REM-Killer bearbeitet wird. Auch Listings in einer Zeitschrift sollten solche Sprünge nicht enthalten, da der eine oder andere Leser die REM-Zeilen wegläßt und nicht alle Sprünge findet, die er zu korrigieren hat.

Auch diese Fehlermeldung kann man »ausblenden«, wenn der Fehler allzu oft auftritt. Drücken Sie dazu die N-Taste.

Die folgende Frage ist wieder so etwas ähnliches. Man kann damit gleich zwei Fehler auf einmal ausblenden: Den Fehler Nr. 7, der angezeigt wird, wenn ein Sprungbefehl auf einen Strukturierbefehl zeigt, den man auch anstelle des Sprungbefehles hätte setzen können. Beispiel:

```
10 A = 4 : GOTO 20
20 RETURN
```

kann man auch einfacher so schreiben:

```
10 A = 4 : RETURN
```

Der zweite Fehler, der von diesem »Schalter« betroffen ist, ist der Fehler Nr. 17. Er wird gemeldet, wenn der Programmierer direkt hinter das

Befehlswort THEN einen GOTO Befehl gesetzt hat (anstelle THEN GOTO 12 schreibt man kürzer THEN 12). Da beides aber Schönheitsfehler sind, kann man sie, wenn sie zu häufig auftreten, beide zusammen ausblenden, indem die Frage mit <J> beantwortet wird.

Die letzte der vier »Ausblendfragen« betrifft gleich drei Fehler: Wie Sie aus der Liste ersehen können, betreffen diese Fehler 12, 13 und 14 Befehlswörter, die ein geübter Programmierer nicht verwendet: LET, NEW und STOP. LET kann man sich sowieso sparen, und NEW und STOP gehören nicht in ein gutes Basicprogramm. Falls der Bediener des BKS anderer Meinung ist, kann er die Ausgabe der drei Fehler unterdrücken, indem er die Frage mit <N> beantwortet.

Als nächstes werden Sie gefragt, ob die fehlerhaften Zeilen auch gelistet werden sollen, oder ob die Ausgabe der Zeilennummer genügt. Beantworten Sie auch diese Frage wieder mit <J> oder <N>.

Die letzte Frage dient dazu, festzulegen, auf welches Gerät die Fehlerliste ausgegeben werden soll. Drücken Sie eine der Tasten <D>, <S> oder <F>:

S: Die Fehlerliste wird auf dem Bildschirm ausgegeben. Dieser Modus ist vor allem bei sehr kurzen Programmen angebracht, und zum Test, ob die ausblendbaren Fehler (siehe oben) zu oft vorkommen.

D: Die Liste wird auf dem Drucker ausgegeben. Die Geräteadresse ist 4, die Sekundäradresse 0. Die Routine wurde für Commodore-kompatible Drucker geschrieben, müßte jedoch auch mit anderen Printern zusammenarbeiten. Diesen Ausgabemodus braucht man, wenn die Fehlerliste schriftlich vorliegen soll, beispielsweise beim Test von Leserlistings. Da die selbe Routine wie zur Bildschirmausgabe verwendet wird, ist die Ausgabe nicht breiter als 40 Zeichen.

Sollten die eckigen Klammern, in denen der Härtecode erscheint (siehe unten), auf Ihrem Drucker nicht oder fehlerhaft (z.B. als deutsche Umlaute) wiedergegeben werden, beachten Sie bitte die untenstehenden Hinweise.

F: Listet die Aufstellung auf Diskette. Dabei wird auf der (eingelegten) Floppy ein Basicprogramm mit dem Namen erzeugt, den Sie nun eingeben. Er beginnt gewöhnlich mit der Kennung »DOC.« für DOCument, diese kann durch Eingabe von <Cursor left> jedoch überschrieben werden (Vorsicht! Den Cursor nicht weiter links als auf das »D« bewegen!). Dieses File kann dann wie ein normales Basicprogramm geladen werden, ansehen können Sie sich die Auswertung durch Eingabe von LIST oder RUN (frei nach Wahl).

Nach der Eingabe dieser Parameter geht das BKS das Basicprogramm nun Zeile für Zeile durch und überprüft, ob es Fehler enthält. Bei jeder neuen Zeile wechselt der Bildschirmrahmen seine Farbe, so erkennen Sie, wenn es länger dauert, daß die Routine nicht abgestürzt ist (das BKS ist so programmiert, daß es nicht abstürzen KANN). Am Ende des Tests werden alle Dateien geschlossen und das BKS beendet.

Jedesmal, wenn ein Fehler auftritt, wird eine Zeile mit folgendem Format ausgegeben (auf Schirm, Drucker oder Diskette):

```
AAAAA: BB[C] blablablabla
```

AAAAA ist die Basic-Zeilennummer, in der der Fehler auftritt. Sie wird rechtsbündig ausgegeben. BB ist die laufende Nummer des Fehlers, siehe dazu

die untenstehende Tabelle.

Es gibt zwei verschiedene Fehlergrade, die C angibt: leichte Fehler, die eigentlich nur Schönheitsfehler sind und keine Fehlfunktion des Programmes bewirken. C ist bei diesen Fehlern 1. Beispiele: überflüssige Spaces, Listschutz, LET-Befehl. Anders ist es bei den schweren Fehlern (C=2): Hier wird sich das Programm mit einer Basic-Fehlermeldung verabschieden, beispielsweise, wenn Sprungbefehle auftreten, die auf nicht vorhandene Zeilen zeigen, wenn das Programm formal zerstört ist (POKEs) oder bei Syntaxfehlern.

Der hier mit »blablablabla« bezeichnete Teil stellt einen Pauschaltext dar, der ungefähr die Art des Fehlers angibt. Die möglichen Texte sind: »FORMATFEHLER«, »SPRUNGFEHLER«, »ÜBERFLÜSSIGER BEFEHL« und »UNERLAUBTER BEFEHL«.

Da dieser Pauschaltext nicht genau die Art des Fehlers angibt, ist der Parameter BB besonders wichtig: Er kann 28 verschiedene Werte annehmen: (in Klammern der Härtecode, Parameter C)

1 {1} Direkt nach der Zeilennummer folgt ein Nullbyte (dies wird zu Listschutzzwecken verwendet).
2 {1} Im Programmtext kommt ein überflüssiges Leerzeichen vor. Nach DATA, in Anführungszeichen oder wenn diese Funktion abgeschaltet ist, werden die Spaces nicht bemängelt.
3 {2} Ein THEN, GOTO, LIST, RUN oder GOSUB Befehl zeigt auf eine nicht existierende Zeile (UNDEF'D STATEMENT)
4 {2} Die hinter einem dieser Befehle stehende Zeilennummer ist größer als 63999.
5 {2} Die hinter einem dieser Befehle stehende Zeilennummer enthält verbotene Zeichen (etwa GOTO 4+6 oder GOTO LABEL). PASCAL gewohnte Programmierer werden diese Funktion des BKS zu schätzen wissen.
6 {2} Eine Basic-Zeile ist länger als 255 Zeichen (Fehler von der Suchroutine).
7 {1} Ein GOTO oder THEN Befehl zeigt auf einen RETURN, GOTO, RUN usw. Befehl (Strukturierbefehl), den man auch einfach anstelle des Sprungbefehles hätte setzen können.
8 {1} Ein Sprungbefehl zeigt auf eine REM- oder Trennzeile (das ist eine Zeile, die nur einen Doppelpunkt enthält). Das kann zu Problemen beim Abtippen führen, wenn die REM Zeile weggelassen wird. Falls in einem Listing dieser Fehler zu oft vorkommt, kann auch die Ausgabe dieses Fehlers abgeschaltet werden.
9 {2} Eine Basic-Zeile ist länger als 255 Zeichen (Fehler von der Hauptroutine). Wenn dieser Fehler auftritt, dürfen eventuelle sonstige Fehler nicht mehr unbedingt ernstgenommen werden, da dann das System vollkommen durcheinandergerät.
10 {2} Ein GOTO oder RUN-Befehl zeigt auf sich selbst (nicht hinter THEN; Beispiel: 10 GOTO 10)
11 {2} Der Befehl CONT darf nicht im Programmtext vorkommen.
12 {1} Der Befehl STOP sollte nicht im Programmtext vorkommen.
13 {1} Der Befehl NEW sollte nicht im Programmtext vorkommen.
14 {1} Der Befehl LET sollte nicht im Programmtext vorkommen.
15 {1} Hinter einem REM Befehl steht ein geshiftetes L (Listschutz).
16 {2} Ein illegales Token (Zeichen, Byte) kommt im Programmtext vor.
17 {1} Der Befehl GOTO sollte nicht direkt hinter THEN stehen, einer von beiden genügt.
18 {2} Hinter einem Befehl fehlt der Parameter (z.B. OPEN).
19 {2} Hinter GO fehlt TO.

20 {1} Hinter GOTO, RUN etc folgen weitere Befehle, die niemals ausgeführt werden (z.B. GOTO 20:PRINT "GEISTERBAHN")
21 {2} Klammer(n) zu viel bzw. wenig.
22 {1} Das Zeichen »^« (Pfeil nach oben) zur Potenzierung sollte vermieden werden (große Rechenungenauigkeit).
23 {2} Der Befehl PRINT# wurde mit ?# abgekürzt. (SYNTAX ERROR)
24 {2} Falsche Reihenfolge der Basiczeilen. Das kann zu Problemen bei Sprungbefehlen führen.
25 {2} Ein falscher Linkpointer kommt im Programmtext vor. (Vor jeder Basic-Zeile steht im Speicher ein Zeiger, der angibt, wo im Speicher die nächste Zeile beginnt. Anhand dieser Zeiger »hangeln« sich u.a. die Sprungbefehle zur gesuchten Zeile.)
26 {2} ON ohne legalen Sprungbefehl.
27 {2} THEN ohne IF.
28 {1} Der Pointer 45/46 zeigt nicht genau auf das Byte hinter dem Basic Programm. Wahrscheinlich ist noch ein Maschinenprogramm angehängt, oder es wurde ein fehlerhafter RENEW-Befehl verwendet.

Die Fehler 6 und 9 unterscheiden sich in ihrer Bedeutung nicht und werden ggf. immer paarweise erscheinen. Diese Meldungen werden innerhalb des BKS an zwei verschiedenen Stellen erzeugt: Nummer 6 in dem Unterprogramm, welches bei Sprungbefehlen die Existenz der angesprochenen Zeile überprüft, und Nummer 9 in der Hauptroutine, die das Basicprogramm Befehl für Befehl durchgeht und nach Fehlern sucht. Die »Fehler«-Meldungen Nr. 11 (CONT) und Nr. 5 (GOTO 4+6) wurden ebenso wie zum Beispiel Nr. 22 oder Nr. 12 nur der Vollständigkeit halber in das BKS aufgenommen. Sie bezeichnen keine schweren Fehler, insofern ist der Härtecode 2 hier prinzipiell übertrieben.

Die eckigen Klammern, in denen der Härtecode erscheint, können auf dem Drucker nur dann richtig wiedergegeben werden, wenn Sie ihn vom deutschen in den amerikanischen Zeichensatz schalten (bei MPS-Druckern nicht nötig). Sollten hier falschen Zeichen ausgegeben werden, und gelingt Ihnen die Anpassung nicht, können Sie den Code auch direkt im Programm ändern, beispielsweise in runde Klammern. Die entsprechenden Speicherzellen sind $ca4e und $ca5f.

Wie funktioniert das BKS intern? Für eine genaue Funktionsbeschreibung reicht hier leider der Platz nicht aus. Der interessierte Leser erhält jedoch vom Autor gegen Einsendung einer Diskette und einer Gebühr von DM 10,- gern den kompletten Quelltext (gilt für alle Programme!). Die Adresse finden Sie in dem im folgenden beschriebenen Programm »BKS.WHAT«.

Dabei handelt es sich um ein wichtiges Zusatzprogramm zum BKS, mit dem die ausgegebenen Listen genau kommentiert werden. Die Anwendung ist einfach: Laden Sie wie oben beschrieben das BKS, dann das zu testende Programm. Führen Sie den Test durch, und lassen Sie sich die Referenz auf den Drucker ausgeben. Danach laden Sie, ohne vorher den Computer auszuschalten, das Programm "BKS.WHAT 5.0" und starten es mit RUN. Sofern vorher eine Auswertung mit dem BKS durchgeführt wurde, erscheint jetzt die Frage nach dem Datum und dem Namen des getesteten Basicprogrammes. Alle anderen Angaben hat das BKS dem Kommentierprogramm bereits übergeben (im »Common-Bereich«, siehe Speicherbelegung). Nach kurzer Zeit wird jetzt ein ausführlicher Kommentar auf den Drucker ausgegeben, und zwar diesmal über die Sekundäradresse 7 (Groß-/Kleinschriftmodus). Darin enthalten ist eine kommentierte Liste ähnlich der oben abgedruckten, die Auskunft über die genaue Bedeutung der aufgetretenen Fehler gibt (siehe Beispielausdruck). Das Druckprogramm wurde für MPS-Drucker geschrieben, und enthält einige Steuercodes dieser Drucker.

Ich wünsche Ihnen nun noch viel Erfolg mit dem BKS!

ANHANG:

Speicherbelegung des BKS (Version 5.0, hexadezimal):

```
0002-0003 temporär (u.a. Zeiger auf Programm)
0334-0344 Filename
C000-CCDD Programm »BKS«
C000-C002 Sprung nach $C687 (Anfang)
C003-C239 verschiedene Texte (gepackt)
C23A-C23B Nummer der gesuchten Basic Zeile für $C37C
C23C      Flag
C23D      Flag: Anführungszeichenmodus
C23E      Flag: REM
C23F-C240 Nummer der aktuellen Zeile
C241      Position in dieser Zeile
C242      Nummer des Fehlers
C243-C246 Integerzahlen für Multiplikation mit 10
C247      aktuelles Token
C248      aktuelle Zeigerposition
C249      Anzahl der Klammerebenen
C24A      Anzahl der Ziffern hinter Sprungbefehl
C24B      Flag: THEN
C24C      Flag: ON
C24D      Geräteadresse für Ausgabe (3, 4, 8)
C24E      Low-Byte der Anzahl der leichten Fehler
C24F      Low-Byte der Anzahl der schweren Fehler
C250      High-Byte der Anzahl der leichten Fehler
C251      High-Byte der Anzahl der schweren Fehler
C252      Flag: DATA
C253      letzter Zustand von $C252
C254      Sekundäradresse für Ausgabe
C255      Hochkommaflag für Listroutine
C256      Flag: Basic Zeilen listen
C257      Pointer für Listroutine
C258      »First«-zeiger
C259-C25A letzte Fehlerzeilennummer
C25B      temporär
C25C      wie $C248
C25D      Flag: ON/Hochkomma
C25E      wie $C248
C25F      letztes Zeichen
C260-C27D Tokentabellen
C27E-C2C1 Texte der Fehler
C2C2-C2F9 Zeiger auf diese Texte
C2FA-C315 Härtecodes der Fehler (AscII)
C316-C37B Vorspann für erzeugtes Basicprogramm
C37C-CCDD 100% Maschinenprogramm. Interessante Routinen:
C37C      Sucht die Zeile (X/Y)
C4DE      Entschlüsselt Text ab (A/Y) und gibt ihn aus
C562      Holt nächstes Zeichen aus Basic Text
C56F      Holt Parameter
C584      Holt Tasten J/N
C687      Anfang des Hauptprogrammes
C6D9      Schleife: Neue Basic Zeile
C710      Hauptschleife: Nächstes Zeichen
C77A      Nächstes Zeichen
```

```
CA0D      REM-Routine
CA19      Unterroutinen zu $CACE
CACE      Gibt Fehlermeldung Nr. X aus
CBE0      fertig
CC5D      Holt Zahl hinter Sprungebefehl
CE00-CEFF Puffer für Listroutine
CF00-CFFF "Common-Bereich" (hier werden die Ergebnisse des BKS an BKS.WHAT
          übergeben)
CF00-CF04 Erkennungstext »NSS88«
CF05      Versionsnummer primär   (=5)
CF06      Versionsnummer sekundär (=0)
CF07      Flag: Fehler Nr. 2     ausgekoppelt (0 = ja)
CF08      Flag: Fehler Nr. 8     ausgekoppelt (0 = ja)
CF09      Identifikationsbyte (123 = BKS läuft, 222 = BKS fertig)
CF0A      Flag: Fehler Nr. 7/17  ausgekoppelt (0 = ja)
CF0B      Flag: Fehler Nr. 12-14 ausgekoppelt (0 = ja)
CF0C-CF27 Tabelle aller 28 Fehler (1 = tritt auf)
```

Anleitung zum Programm »HELP«:

Dieses Programm ist vor allem für Basic-Einsteiger gedacht, die oft vor dem Bildschirm sitzen und sich wundern, was der Commodore 64 doch alles für Fehlermeldungen erzeugen kann, die er selbst nicht einmal aus dem Handbuch kennt. Wie schön wäre es doch, wenn der Computer auf Anforderung die soeben gezeigte Meldung kurz mal erklären könnte.

Genau dies ist mit dem Programm »HELP« in Zukunft problemlos möglich. Laden Sie das Programm mit dem Befehl

LOAD "HELP !!",8

und starten durch Eingabe von RUN. Es wird nun noch ein Maschinenprogramm nachgeladen und gestartet. Ab jetzt haben Sie die Möglichkeit, nach jeder Fehlermeldung, die der Computer ausgibt (vom einfachen SYNTAX ERROR über den TYPE MISMATCH ERROR oder ILLEGAL DIRECT ERROR bis hin zum unverständlichen FORMULA TOO COMPLEX ERROR) einfach den neuen Befehl

HELP

einzugeben (und <RETURN>), und schon erklärt der Computer in ein paar Zeilen auf Deutsch, was der Grund für diese Fehlermeldung sein könnte. Der HELP-Befehl soll nur im Direktmodus und direkt nach der Ausgabe der Meldung, nicht erst nach dem nächsten (möglicherweise ja wieder korrekten) Befehl verwendet werden.

Nach einem <RESET> oder versehentlichen Abschalten der Erweiterung durch ein anderes Programm kann HELP, so es sich noch im Speicher befindet, mit

SYS 29952

wieder gestartet werden.

Dieses kleine Hilfsprogramm (der Hauptteil des Nachladeprogrammes wird von den erklärenden Texten belegt) ist sicher eine nützliche Hilfe beim Programmieren, die vor allem Einsteiger sehr bald schätzen werden.

HIlfe - die Datas kommen!

Dieses »clevere« Utility ist vor allem dann praktisch, wenn Sie mit DATAs arbeiten. Es baut einige neue Befehle in den C 64 ein, die beim Programmieren oder Eintippen von DATAs nützlich sind. Obwohl es vollkommen in Maschinensprache geschrieben ist, müssen Sie nicht in Maschinensprache programmieren können, um es anzuwenden.

Zahlen in einer FOR..NEXT-Schleife mit READ aus DATA-Zeilen zu lesen und diese dann in den Speicher zu POKEn ist die am häufigsten angewandte Methode, zum Beispiel ein Assemblerprogramm oder Sprite-Daten in den Speicher zu laden. »DATA-Aid« kann Ihnen in vielfältiger Hinsicht helfen, wenn Sie es mit DATA-Daten zu tun haben.

Sie laden dieses in Maschinensprache verfaßte Utility wie ein Basicprogramm mit

```
LOAD "DATA AID",8
```

und starten mit

```
RUN
```

Jetzt wird automatisch eine Routine in einen vor Basic geschützten Bereich kopiert und die Erweiterung installiert. Der Bildschirm färbt sich grün und eine Meldung erscheint. Jetzt wurden fünf neue Befehle ins Basic eingebaut. Diese können wie normale Basicbefehle verwendet werden, im Programm- oder Direktmodus. Das Ausrufezeichen dient als Kennzeichen dafür, daß ein neuer Befehl folgt.

Wenn Sie jetzt gleich mit DATA-AID arbeiten möchten, sollten Sie zunächst mit

```
NEW
```

den Basicspeicher löschen, da hier ja noch die SYS-Zeile zum Start von DATA-AID steht. Und das sind die neuen Kommandos:

!R - Restore. Dieser Befehl setzt den DATA-Zeiger auf eine bestimmte Programmzeile. !R wirkt also wie RESTORE, nur daß nicht auf den Anfang des Programmes gesetzt wird, sondern bei

```
!R 100
```

zum Beispiel auf die DATA-Zeile 100 (falls es diese gibt, sonst auf die nächste DATA-Zeile). Sie können, ähnlich wie beim RENUMBER n Befehl des C 128, nicht nur Zahlen, sondern beliebige numerische Ausdrücke hinter !R verwenden, also zum Beispiel auch

```
!R 9*X+40-PEEK(3)
```

Der nächste Befehl heißt

!S - Speed Poke. Dieser Befehl tranportiert sehr schnell Zahlen aus DATA-Zeilen in den Speicher. Der Befehl lautet genau:

!S Zeilennummer, Speicherstelle

Also schafft zum Beispiel

!S 100,8192

alle Zahlen, die in den DATA-Zeilen 100ff abgelegt sind, ab 8192 in den Speicher. Der Vorgang endet, wenn eine negative Zahl gelesen wird. Beispiel: Dieses Programm schreibt DATA-AID rechts oben auf den Bildschirm:

```
10 !S20,1024
20 DATA68,65,84,65,45,65,73,68,-1
30 !S40,55296
40 DATA1,1,1,1,1,1,1,1,-1
50 END
```

Diese Methode arbeitet über viermal so schnell wie die bekannte Befehlsfolge

```
10 READA:IFA<0THENRETURN
20 POKEM,A:M=M+1:GOTO10
```

!S kann in Programmen oder im Direktmodus verwendet werden. Verwenden Sie diesen Befehl nur, wenn die gelesenen DATA-Zeilen nur Werte von 0 bis 255 enthalten, andernfalls erhalten Sie eine Fehlermeldung. Haben Sie die negative Zahl am Ende vergessen, erscheint der normale ?OUT OF DATA ERROR. Dieser negative Wert wird natürlich nicht in den Speicher geschrieben, er markiert nur das Ende der Liste. Übrigens brauchen Sie gar keine echte negative Zahl zu setzen, es reicht im Prinzip das Minuszeichen.

!C - Check. Dieser Befehl arbeitet grundsätzlich wie !S, allerdings werden die Werte nicht in den Speicher geschrieben, sondern einfach addiert. Die »Prüfsumme« wird ausgegeben. So können Sie testen, ob die Werte in Ordnung sind. Daher entfällt hier auch die Angabe der Speicherzelle, der Befehl lautet also zum Beispiel !C 100, wenn die DATA-Zeilen ab 100 summiert werden sollen. Hier können auch positive Zahlen über 255 verwendet werden. Der Computer gibt die Summe in Form einer Meldung DIE PRUEFSUMME IST XXXXX aus.

!D - DATA-Zeilen erzeugen. Benutzen Sie diesen Befehl, wenn schon Daten im Speicher stehen, und Sie diese in DATA-Zeilen verwandeln wollen. Dieser Befehl hat folgende Syntax:

!D ADR1,ADR2+1,ZEIL,ANZ,STEP

Dabei gibt ADR1 und ADR2 den Speicherbereich an (bitte denken Sie daran: Die Endadresse ADR2 muß um eins erhöht angegeben werden), ZEIL ist die erste Zeilennummer, die dann im Schritt STEP erhöht wird (z.B. 10). Dabei werden ANZ DATAs pro Zeile untergebracht. ANZ ist aus dem Bereich von 1 bis 19. So wandelt zum Beispiel der Befehl

!D 52525,53248,2000,19,1

das Programm DATA-Aid (52525 bis 53247) in DATA-Zeilen ab 2000 in der

Schrittweite 1, es werden 19 DATAs pro Zeile ausgegeben.

Die mit diesem Kommando erzeugten Zeilen könnten dann zum Beispiel mit einem Speed-Poke (!S) wieder eingelesen werden. Bedenken Sie aber bei Verwendung von !D, daß die DATA-Zeilen etwa viermal so viel Speicherplatz belegen wie der ursprüngliche Speicherbereich. Wenn ein ?OUT OF MEMORY ERROR zu sehen ist, müssen Sie den Rechner aus- und wieder einschalten, damit Sie weiterarbeiten können. Auch wenn der Befehl !D besonders bei größeren Speicherbereichen einige Zeit in Anspruch nimmt, während der sich der Computer nicht zurückmeldet (am Ende erscheint dann READY.), dürfen Sie den Vorgang auf keinen Fall mit <RUN STOP/RESTORE> abbrechen. Die neuen DATA-Zeilen werden übrigens einfach an der korrekten Stelle in ein ggf. schon bestehendes Basicprogramm eingefügt, also als ob Sie sie neu über Tastatur eingegeben hätten. Bereits bestehende Basiczeilen mit gleichen Nummern wie die neuen DATA-Zeilen werden überschrieben. !D ist in erster Linie für die Verwendung im Direktmodus geschaffen, kann aber auch vom Programm aus verwendet werden. Nach der Ausführung befinden Sie sich allerdings auf alle Fälle im Direktmodus.

!Z - Exit. Dieser Befehl schaltet die Erweiterung ab. Dies sollten Sie tun, wenn Sie andere Erweiterungen verwenden wollen, die den selben Speicherbereich belegen oder ebenfalls neue Befehle definieren. Sofern sich DATA-Aid noch im Speicher befinden, kann es mit

```
SYS 52525
```

(leicht zu merken) jederzeit wieder aktiviert werden.

Eine Bemerkung noch zu den neuen Befehlen: Werden sie direkt hinter THEN verwendet, müssen Sie sie mit einem Doppelpunkt abtrennen, also zum Beispiel so:

```
IF A > 40 THEN : !R A*10
```

Wenn Sie DATA-AID testen wollen, schreiben Sie einfach große Teile des Bildschirms mit beliebigem Text voll. Diesen Screen wollen wir in DATAs im Programm festhalten. Dazu geben Sie jetzt die Befehle

```
NEW

!D 1024,2024,2000,19,2
```

ein und gedulden sich einige Sekunden. Nach Erscheinen der READY.-Meldung geben Sie LIST ein, um das Ergebnis zu begutachten: Scheinbar endlose Kolonnen von DATA-Zeilen, beginnend bei Zeile 1000. Fügen Sie selbst noch die Zeile hinzu:

```
9999 DATA-1
```

Jetzt löschen Sie den Bildschirm und holen den Inhalt wieder zurück, natürlich mit Speed-Poke:

```
!S 2000,1024
```

erledigt das für uns. Anmerkung: Auf älteren Modellen des C 64 ist der Bildschirminhalt noch nicht sichtbar, da die Textzeichen die Farbe des Hintergrundes haben. Um sie sichtbar zu machen, verändern Sie einfach nachträglich die Farbe des Hintergrundes. Ist der Bildschirm momentan zum

Beispiel nicht weiß, machen Sie ihn mit

POKE 53281,1

weiß.

Nun für den interessierten Leser noch eine kurze Beschreibung der programminternen Funktionsweise: DATA-Aid liegt im Speicherbereich 52525 bis 53247 ($CD2D-$CFFF). In der Zeropage werden die sonst unbenutzten Zellen 2 bis 5, 166/7, 181/2, 251-3 und 704-8 verwendet. Die neuen Befehle werden definiert, indem der IGONE-Vektor (776/7) auf eine neue Routine verbogen wird, die das Ausrufezeichen sucht.

Der !R Befehl berechnet die Adresse der angegebenen Zeilennummer (Routine ab $A613) und verbiegt den DATA-Vektor (65/66) auf die Adresse. Die Befehle !S und !C rufen vor ihrer eigentlichen Arbeit !R auf, um die richtige DATA-Zeile »anzufahren«. Der Basic-Pointer 122/3 wird auf den Stack gerettet und auf die Adresse der ersten DATA-Zeile verbogen. Der C 64 sucht das Befehlstoken 131 (DATA) und setzt den Zeiger dahinter, auf den ersten DATA-Wert. Nun können mit den normalen FRMNUM-Routinen ($AD8A, $B79E) die Werte gelesen und in den Speicher geschrieben bzw. addiert werden. Die Addition bei !C geschieht mit Hilfe der Floatingpoint-Arithmetik, dabei wird der FAC immer wieder bei 704-708 zwischengespeichert.

Der Befehl !D verbiegt den Vektor zur Eingabe einer Basiczeile (770/1) auf eine eigene Routine. Ab $A480 springt der Computer über diesen Vektor. Er arbeitet nun nicht die Routine ab $A560 ab, die von Tastatur eine Zeile in den Basic-Eingabepuffer ab 512 holt, sondern eine DATA-AID-Routine, die auch im Basic-Eingabepuffer eine Zeile der Form Zeilennummer, DATA, Werte durch Kommata getrennt aufbaut. Diese Zeile wird dann der normalen Routine ab $A486 übergeben, diese wertet die Zeile aus und baut sie in das Basicprogramm ein.

Wir wünschen Ihnen viel Spaß und Erfolg mit DATA-AID. Genießen Sie die Vorzüge und den Komfort, den Ihnen dieses kurze und neuartige durchdachte Tool bietet!

TI und TI$ greifen auf die CIA-Uhr zu

Sinn und Zweck der Routine »TI$« ist, die Arbeitsweise der reservierten Basic-Variablen TI und TI$ zu ändern. Bisher wird der System-Interrupt-Zähler abgefragt bzw. gesetzt, wenn man mit TI oder TI$ arbeitet. Da dieser sehr ungenau ist (Abweichungen bis zu einer halben Stunde per Tag sind keine Seltenheit) und bei Ein/Ausgabe sogar angehalten wird, bietet es sich an, stattdessen die Echtzeituhr einer CIA zu verwenden. Genau das erfolgt mit der nur wenige Bytes kurzen Routine TI$. Wurde sie erst einmal mit

```
LOAD "TI$",8
RUN
```

aktiviert, baut das Programm intern die TI$ und TI um. Die Funktionen sind voll kompatibel zur Originalversion (eine Ausnahme gibt's, siehe unten). Bevor Sie TI$ oder TI abfragen können, muß allerdings erst eine Uhrzeit

gesetzt werden, da die Uhr sonst noch steht. Dazu verwenden Sie den normalen Befehl

TI$ = "HHMMSS"

Drei Zugriffsarten sind erlaubt. TI$ kann sowohl gelesen wie auch geschrieben werden, und TI kann nur gelesen werden. Dies entspricht dem Original-System. Eine Zuweisung wie

TI = 12345

führt zur Fehlermeldung

?CAN'T DEFINE ERROR

die neu eingeführt wurde. Wichtiger Hinweis: Soll eine Zuweisung der Form TI$=... direkt nach THEN erfolgen, muß nach THEN ein Doppelpunkt stehen. Also nicht

IF A = 10 THEN TI$ = "230000"

sondern

IF A = 10 THEN : TI$ = "230000"

Dies ist die einzige Änderung zum Originalsystem. Vergessen Sie den Doppelpunkt, wandert die Neueinstellung ins Nirwana (sog. »Ablage P«). Ansonsten können Sie mit TI$ und TI operieren, wie Sie es gewohnt sind. Im Direkt- oder Programm-Modus. Also auch als Bestandteile eines Terms (MID$(TI$,3,2) ermittelt die Minuten). Nur daß eben nicht die Systemuhr, sondern die Uhr der CIA 1 ($DC00) Verwendung findet. Obwohl die Uhr der CIA eine Genauigkeit von einer Zehntelsekunde hat, erscheint bei TI$ nach wie vor nur ein String mit sechs Zeichen, um die Kompatibilität zu wahren. Die 1/10 Sekunden kann der interessierte Anwender über TI ermitteln. TI hat jetzt natürlich nur noch eine Auflösung von 1/10 Sekunde, gibt aber - Kompatibilität - dennoch nach wie vorher die Zeit in 1/60 Sekunden an (der Wert von TI ist demnach immer ein Vielfaches von 6). Änderungen an TI$ bewirken automatisch Änderungen an TI. Die Zuweisung TI$ = ... kann auch nach LET stehen. Die Variable TI% kann weiterhin als »stinknormale« Integer-Basicvariable Verwendung finden, die nichts mit der Uhrzeit zu tun hat. Nach TI können aber auch beliebig weitere Buchstaben oder Ziffern folgen, wie es sich für einen Variablennamen gehört (also TIME$="123456" oder PRINT TIIII/60 "SEKUNDEN"). Außerdem arbeitet natürlich auch unsere neue TI$-Version mit einer 24-Stunden Uhr, obwohl die CIA-Uhr eigentlich nur 12 Stunden (AM/PM-Flag) läuft. Umrechnungsroutinen sind enthalten. Wie Sie sehen, steckt viel Liebe im Detail dieser Erweiterung. Das alles, um die Routine voll kompatibel zu machen. Selbstverständlich wird die CIA-Uhr vor dem Betrieb auf 50 Hertz geschaltet, sonst würde unsere Uhr, die ihren Takt aus der Netzfrequenz bezieht, grob falsch gehen.

Eine zweite neue Fehlermeldung gibt es noch:

?TIME FORMAT ERROR

erscheint immer dann, wenn der String, der an TI$ übergeben wird, keine vernünftige Uhrzeit enthält. Mögliche U(h)rsachen: Länge ungleich 6 Bytes, enthält nicht nur Ziffern, Minuten oder Sekunden größer 59, Stunden größer 23.

Die Routine kann auch wieder abgeschaltet werden, dazu geben Sie einfach

SYS 58451

ein. Wiedereinschalten ist, ggf. auch nach einem Reset, mit

SYS 49152

wieder möglich.

Die Funktionsweise ist einfach zu verstehen. Für diejenigen, die das interessiert, wurde das Assemblerlisting (Profi-Ass Format) sehr gut kommentiert. Daher sollen hier nur allgemeine Hinweise stehen. Der Computer verbiegt die Vektoren zur Ausführung eines Basic-Befehles und zur Auswertung eines Basic-Arguments auf neue Routinen. Die neue Befehlsroutine (Label NEUBEF) prüft, ob eine Zuweisung an die Variable TI$ erfolgt (bei TI=... wird die Fehlermeldung ausgegeben). Wenn ja, wertet der C 64 den String aus, berechnet die Uhrzeit und schreibt sie in die Register der CIA. Die Zehntelsekunden werden auf Null gesetzt. Als »Bonbon« erkennt die Routine auch eine Zuweisung der Form

LET TI$ = "123456"

und bearbeitet sie korrekt. Andere Variablen als TI und TI$ werden normal in der alten Routine bearbeitet.

Die neue Routine zur Auswertung eines Terms prüft auf Vorliegen von TI oder TI$. TI% wird von der alten Routine bearbeitet. In beiden Fällen wird jetzt erst einmal die Uhr der CIA 1 ausgelesen und in einen sechs Bytes langen String HHMMSS ab Adresse 255 codiert. Die Zehntelsekunden werden für TI separat erfaßt. Bei TI$ ist die Arbeit schon getan, der String wird zurückgegeben. Dazu dient eine Unterroutine von STR$. Bei TI wird's jetzt sehr kompliziert. Die einzelnen Stellen von TI$ werden von links kaskadisch mit Vielfachen von 60 multipliziert und addiert. Dadurch kommt die Uhrzeit in Sekunden zustande, allerdings ohne Zehntelsekunden. Das Ergebnis multiplizieren wir noch einmal mit 60 und addieren die Zehntelsekunden mal 6 dazu. Das Ergebnis ist TI, die Zeit in 1/60 Sekunden mit einer Genauigkeit von 1/10 Sekunde.

Die neue Routine benutzt fünf normalerweise unbenutzte Speicherzellen zur Datenspeicherung: 155 und 247 bis 250. Das Programm liegt im Bereich von 49152 bis 49799 jeweils einschließlich. Weiterer Speicher wird nicht belegt. Im Maschinenprogramm muß an zwei Stellen im Dezimalmodus gerechnet werden (SED). Davor wird jeweils der Interrupt ausgeblendet, damit der Rechner nicht abstürzt, wenn der Controller bei gesetztem D-Flag den IRQ auslöst (wird von vielen Programmierern, auch denen des 1541-Systems, falsch gemacht).

Sollte es die spezielle Anwendung erfordern, kann das Programm leicht auf die Uhr der CIA 2 umgestellt werden. Dazu ändern Sie im Quelltext in Zeile 130 das Label CIA von $DC00 in $DD00.

Nun bleibt noch eine Frage. Sie haben ja gar keinen Beweis, daß der Computer mit der neuen Routine tatsächlich nicht mehr auf die Systemzeit zugreift, sondern auf die Uhr der CIA. Die Wirkung ist ja haargenau die selbe. Dazu können Sie mit einem Programm, das die CIA-Zeit am Bildschirm einblendet, Kontrollen vornehmen (Beispiel: »Superinfoirq«).

Anleitung zum »REM-KILLER ++«

Dieser besondere REM-Killer unterscheidet sich wesentlich von herkömmlichen Programmen dieser Art. Während solche Tools (z.B. Trick des Monats in 64'er 01/90) gewöhnlich nur die REM-Zeilen aus einem Basicprogramm entfernen, denkt dieses Tool mit und verändert auch zum Beispiel die GOTO-Befehle im übrigen Programm so, daß es hinterher immer noch läuft. Obwohl dieses Hilfsprogramm aus Geschwindigkeits- und Komfortgründen vollkommen in Maschinensprache geschrieben ist, kann es wie ein Basicprogramm geladen und gestartet werden. Es installiert sich dann im Speicher und wartet darauf, mit einer besonderen Tastenkombination aktiviert zu werden.

Zum Beispiel vorher:

```
10 PROGRAMMTEIL 1
20 IF A = 5 GOTO 40
30 END
40 REM ALTERNATIVE
50 PRINT "..."
60 STOP:REM DAS ENDE
```

Nach der Behandlung mit dem neuartigen REM-Killer sind nicht nur die REM-Zeile 40, die ja nur unnötig Speicherplatz kostet, und der REM-Befehl in Zeile 60 verschwunden, sondern es wurde auch gleich der »verkappte« GOTO-Befehl in Zeile 20 so verändert, daß die Wirkung nachher die gleiche ist:

```
10 PROGRAMMTEIL 1
20 IF A = 5 GOTO 50
30 END
50 PRINT "..."
60 STOP
```

Ohne die Änderung der Zeile 20 hätte sonst ein ?UNDEF'D STATEMENT ERROR IN 20 auftreten können, da der GOTO-Befehl auf eine nicht mehr existierende Zeile gewiesen hätte.

Die Anwendung ist ganz einfach: Laden Sie zunächst das Tool mit dem Befehl

```
LOAD "REM-KILLER++",8
RUN
```

Es erscheint eine Einschaltmeldung. Dann laden Sie das Basicprogramm ganz normal, das gekürzt werden soll. Der Basicspeicher sollte - wie üblich - bei $801 liegen. Dies ist im normalen Betrieb des C 64 der Fall. Zum Kürzen starten Sie nun den Rem-Killer, indem Sie die beiden SHIFT-Tasten gleichzeitig drücken. Wieder erscheint eine Bestätigung.

Es wird nun die Programmlänge in Byte ausgegeben, die Kürzung erfolgt, die Befehle GOTO, GOSUB, THEN sowie ON werden automatisch angepaßt, und das Tool gibt die neue Länge aus. Das gekürzte Programm kann jetzt ganz normal weiterbearbeitet oder gespeichert werden. Das Tool erkennt automatisch, wenn sich kein Programm im Speicher befindet, oder das im Speicher befindliche Programm nur aus REM-Zeilen besteht.

Kurz zur Funktionsweise: Dieses Tool geht das Basicprogramm in drei »Passes« durch. Im ersten Durchgang (Pass 1) werden alle REM-Befehle erkannt und in Pass 2 aus dem Programm entfernt. Dabei legt das Utility eine Tabelle mit allen im Programm vorkommenden Basic-Zeilennummern an. Hier wird jeder Zeilennummer des Ursprungsprogrammes eine Nummer im neuen Programm zugewiesen, im obigen Beispiel etwa:

10 -> 10
20 -> 20
30 -> 30
40 -> 50 (da Zeile 40 nicht mehr existiert)
50 -> 50
60 -> 60

Im dritten Durchgang sucht der REM-Killer im Programm nach den Befehlen GOTO, GOSUB und THEN mit einer oder mehreren durch Komma getrennten Zeilennummern (für den ON-Befehl) und korrigiert die Nummern nach der Tabelle. Sollte beispielsweise aus einem GOTO 90 ein GOTO 100 werden, da Zeile 90 eine REM-Zeile war, wird das Basicprogramm automatisch an dieser Stelle um ein Byte verlängert, damit die Zahl 100 hinter den GOTO-Befehl paßt.

Bei Laden und Starten des Utilities mit RUN wird der IRQ-Vektor auf eine neue Routine verbogen, die die beiden SHIFT-Tasten prüft. Dazu wird 60 mal in der Sekunde direkt mit Hilfe der CIA-Register $DC00 und $DC01 die Tastatur-Matrix abgefragt. Wurde <SHIFT/SHIFT> gedrückt (wahlweise auch die SHIFT-LOCK Taste zusammen mit der rechten SHIFT-Taste), schreibt das Programm einen Befehl in den Tastaturpuffer:

SYS 49155

Mit diesem Befehl wird der eigentliche REM-Killer gestartet. Sie können diesen Befehl natürlich auch ganz normal im Direktmodus eingeben, wenn Ihnen das komfortabler erscheint. Nach einem Ausstieg mit Reset kann des Tool, sofern es sich noch im Speicher befindet, mit SYS 49152 wieder gestartet werden. Dann ist wieder die Tastenkombination aktiv.

64 Keys: Befehle auf Tastendruck

Wer sagt denn, daß man jedesmal bei der Basicprogrammierung mühsam die Schlüsselwörter von Hand eingeben muß? Sie brauchen nur die CTRL-Taste mit einer anderen Taste zu drücken, und schon erscheint ein vollständiger Basicbefehl am Bildschirm. Geben Sie öfters Listings aus Zeitschriften ein? Dann haben Sie auf 64 Keys gewartet!

Laden Sie das Programm mit LOAD "64 KEYS (LADER)",8 und starten es mit RUN. Obwohl die Erweiterung in reiner Maschinensprache programmiert wurde, kann das Programmfile wie ein Basicprogramm geladen, gestartet, gespeichert und kopiert werden. Die Routine wird nun in einen Speicherbereich ab 49152 verschoben und dort gestartet. Ein Titelbild erscheint. Der Rechner arbeitet scheinbar ganz normal weiter. Drücken Sie jetzt aber einmal <CTRL> und gleichzeitig eine Buchstabentaste: Voila - ein Befehl erscheint, als ob Sie ihn ganz normal über Tastatur eingegeben hätten.

Sie können die Schlüsselwörter nach wie vor voll ausschreiben, müssen das aber nicht unbedingt. Es ist nur dann notwendig, wenn der gewünschte Befehl nicht in der untenstehenden Tabelle vorkommt. Innerhalb des Quote- oder Insert-Modus (erster wird eingeschaltet, wenn Sie ein Anführungszeichen eingeben, letzter, wenn Sie <SHIFT-INST/DEL> drücken) reagiert die Erweiterung nicht.

Nach Reset oder <RUN/STOP-RESTORE> schalten Sie 64 Keys, so es sich noch im Speicher befindet, mit

SYS 49152
wieder ein.

Im einzelnen werden folgende Schlüsselwörter wiedergegeben, wenn Sie die angegebene Taste mit <CTRL> drücken:

A	ASC	B	STEP	C	CHR$
D	DIM	E	END	F	FOR
G	GET	H	STOP	I	INPUT
J	GOTO	K	GOSUB	L	LEFT$
M	MID$	N	NEXT	O	OPEN
P	POKE	Q	PEEK	R	RIGHT$
S	STR$	T	TAB(	U	USR
V	VAL	X	READ	W	WAIT
Y	RESTORE	Z	SYS	=	LIST
.	STEP	,	THEN	?	PRINT
<-	DATA	*	REM	-	NEW
+	RUN	^	CONT	£	VERIFY
:	RND	;	FRE	f1	INT
f3	ABS	f5	SGN	f7	LEN
RETURN	RETURN	CLR	CLR	CRSR left	SAVE
CRSR down	LOAD	INST	IF	Klammeraffe	CLOSE

Es sind also alle Tasten des C 64 belegt, natürlich mit Ausnahme der RESTORE-Taste, der Steuertasten <SHIFT>, <RUN/STOP> und <CBM> und der Zifferntasten, da diese mit <CTRL> die Farbumschaltung erledigen.

Kurz noch zur Funktionsweise: Da 64 Keys nicht den IRQ-Vektor verbiegt, ist die Erweiterung sehr kompatibel zu sonstigen Zusatzprogrammen, die nicht den Bereich von 49152 bis 49448 belegen. Hier wird vielmehr der Key-Vektor (655/656) auf eine eigene Routine verbogen.

Viel Spaß mit 64 Keys. Genießen Sie den Komfort, den dieses sehr kurze Utility Ihnen bietet!

Anleitung zu »Array of Byte«

Vor allem Pascal-gewohnte Programmierer kennen die Array-of-Byte-Funktion, mit der sich der gesamte Speicher des Computers wie eine indizierte Variable ansprechen läßt. Unsere kleine Erweiterung bringt auch dem C 64 diese nützlichen Befehle bei. Endlich wird PEEK und POKE überflüssig!

Die Erweiterung »Array of Byte« ist vor allem für Programmierer nützlich,

die ihre Programme übersichtlicher und leichter verständlich schreiben möchten. Fand sich bisher in fast jedem längeren Basicprogramm ein Wust aus PEEK- und POKE-Befehlen, um bestimmte Spezialeffekte durch direkte Speichermanipulation zu erreichen, kann man mit der hier vorgestellten Erweiterung alle 65536 Speicherzellen, die der 64'er bietet, wie eine riesige indizierte Variable ansprechen (indizierte Variablen sind Variablen mit einem Parameter in Klammern hinter dem Variablennamen). Dieser Vorzug war bisher verschiedenen Pascal-Systemen vorbehalten.

Obwohl die Erweiterung aus Komfort- und Geschwindigkeitsgründen vollständig in Maschinensprache geschrieben werden, kann sie dank des Ladeprogramms wie ganz normales Basic geladen und gestartet werden:

```
LOAD "ARRAY OF BYTE",8
RUN
```

Nach kurzer Zeit erscheint eine knappe Anleitung auf dem Bildschirm. Jetzt stehen ein neues Schlüsselwort zur Verfügung: MEM (xxx). Dabei gibt xxx (ganzzahlig mit 0 <= xxx < 65536) die Nummer der anzusprechenden Speicherzelle an. Dieser Ausdruck kann wie normale Arrays in Befehlen verwendet werden und als Parameter in Termen stehen, beispielsweise so:

```
MEM (53280) = 0 wirkt wie POKE 53280,0
PRINT MEM (144) wirkt wie PRINT PEEK (144)
MEM (53269) = MEM (53269) OR 16 (schaltet Sprite 4 an)
IF MEM (808) <> 237 THEN GOTO 900
```

und so weiter. Es gibt nur zwei Einschränkungen. Hinter LET darf MEM nicht für die Zuweisung verwendet werden.

```
LET MEM (53280) = 1 ist verboten,
LET A = MEM (700) ist jedoch erlaubt.
```

Außerdem sollte nach THEN ein Doppelpunkt folgen, falls der neue MEM-Befehl zuweisend direkt dahinter steht. Also nicht

```
IF A = 400 THEN MEM (400) = 60
```

sondern

```
IF A = 400 THEN : MEM (400) = 60
```

Diese beiden Einschränkungen haben programmtechnische Gründe und wiegen in der Praxis nicht schwer. Außerdem darf einer Speicherzelle MEM natürlich nur ein ganzzahliger Wert von 0 bis 255 zugewiesen werden. Die Verwendung der Variablen MEM oder ME (...) in Basicprogramme als normales Array ist verständlicherweise bei aktiviertem Array of Byte nicht möglich. Es sollte noch angemerkt werden, daß Sie vor dem Start eines (fertigen) Basicprogramms mit den neuen Befehlen erst die Erweiterung »Array of Byte« wie beschrieben laden und starten müssen.

Anleitung zur INFORM-Routine: Komfortable Eingabe

Der INPUT-Befehl des C 64 hat einige entscheidende Nachteile: Mit den Cursortasten etwa kann ohne weiteres die Bildschirmmaske zerstört werden. Leider ist es auch nicht möglich, eine maximale Eingabelänge vorzugeben oder Sonderzeichen wie Kommas, Doppelpunkte oder dergleichen mehr einzugeben. Manchmal benötigt man INPUT im Direktmodus, aber es geht nicht! INFORM ist eine 143 Bytes kurze Ersatzroutine, die alle Nachteile beseitigt. Im Gegensatz zu anderen einschlägigen Lösungen dieser Art wird die Eingabe nicht einfach im Speicher abgelegt, sondern wie bei INPUT einer Basic-Variablen zugewiesen.

Das Programm wird mit

```
LOAD "INFORM 49152",8,8
```

geladen. Danach soll NEW eingegeben werden (entfällt, falls INFORM in einem eigenen Programm nachgeladen wird). Jetzt können Sie, sogar im Direktmodus, die Routine mit

```
SYS 49152, LEN, VAR$
```

aufrufen. LEN gibt an, wie viele Zeichen die Eingabe maximal umfassen darf. Setzen Sie LEN auf Null, können nur Leerstrings eingegeben werden. LEN muß kleiner als 256 (ein Byte) sein. Für LEN können Sie Ziffern einsetzen, aber auch numerische Terme aus Variablen und Rechenzeichen.

VAR$ ist die Stringvariable, der die Eingabe zugeteilt werden soll. Einige Beispiele für erlaubte Befehle:

```
SYS 49152, 2, A$
SYS 49152, 56-15, TZ$
SYS 49152, LEN(A$), A$
SYS 49152, PEEK(90)-DD/2, DM$
```

und so weiter.

Bei INFORM können Korrekturen am eingegebenen Text nur mit der DEL-Taste vorgenommen werden, die das zuletzt eingegebene Zeichen löscht. Alle Cursortasten sind gesperrt. Mit <RETURN> schließen Sie die Eingabe ab.

Eine kleine Schwäche: Mit INFORM kann pro Wurf immer nur eine Variable definiert werden (die übrigens, wie sich das für eine ordentliche INPUT-Alternative gehört, angelegt wird, falls sie vorher noch nicht definiert war). Mehrfacheingaben wie bei

```
INPUT "NAME, ALTER"; NA$,AL
```

sind nicht möglich. Dieser Nachteil ist die »Schattenseite« des Vorteils, daß jetzt auch Doppelpunkte, Kommas und Anführungszeichen in der Eingabe enthalten sein dürfen.

Interessant an dieser Routine ist vor allem der Teil ab $c02d, der die Eingabe aus dem Kassettenpuffer liest und in der Stringvariable speichert. Es handelt sich um eine kürzere, einfachere und sicherere Alternative zu der von der 64'er Redaktion in 64'er 9/90 Seite 56 im Leserforum vorgestellten Möglichkeit. Falls Sie das Programm verändern oder in eigene Projekte einbauen wollen, müssen Sie nur beachten, daß die Adresse, an der der JSR SUCHVAR-Befehl (momentan an $c008) beginnt, nicht das Lowbyte $28 (dez. 40) hat (siehe C 64-Betriebssystem ab $b11d und $af28). Dieses Problem tritt in

der Praxis jedoch kaum auf und hat keine Bedeutung für reine Anwender von INFORM, es soll hier nur der Vollständigkeit halber erwähnt werden.

Dem fortgeschrittenen Leser wird es sicher nicht schwerfallen, INFORM in eigene Programme oder Erweiterungen einzubinden. Eine Hilfe mag der ausführlich kommentierte Quelltext sein.

Die Telefonbuchsuche: »GENIE-SUCH«

Eine spezielle Maschinensprache-Routine, die in einem String-Array blitzschnell beliebige Texte findet. Das Besondere: Es erfolgt kein einfacher 1:1 Vergleich, sondern die Routine drückt auf Wunsch ein »Auge zu«, wenn der Suchstring nur fast, nicht aber exakt gefunden wird. Nebenbei erhalten Sie auch eine blitzschnelle Suchroutine, die stundenlange Basic-Schleifen endgültig überflüssig macht.

Nehmen wir einmal an, Sie wollen ein Telefonbuch-Programm schreiben. Dazu würden Sie in je einem Variablen-Array (indizierte Variable) die Namen der Personen und die Telefonnummern speichern. Jetzt wird der Name vom Anwender eingegeben, und eine Suchroutine tritt in Aktion, die jedes Element des Namensfeldes mit der Eingabe vergleicht. Bei Übereinstimmung wird die entsprechende Telefonnummer ausgegeben, sonst eine Fehlermeldung.

Wenn Sie jedoch zum Beispiel statt des gespeicherten »MEIER« auf der Tastatur »MAIER« eingeben, muß das Programm passen. Um es etwas bedienerfreundlicher zu gestalten, wäre eine Suchroutine praktisch, die kleine Tippfehler oder Ungenauigkeiten im Suchstring automatisch korrigiert. Eine solche Routine ist die hier vorgestellte Routine »GENIE-SUCH«. Sie heißt so, weil sie bei der Suche nach einem genialen System vorgeht, um auch Tippfehler wie »Dreher« (»HBUER« statt »HUBER«), falsche Buchstaben (»HUPER« statt »HUBER«), Vertauschungen in der Groß/Kleinschrift oder zu lange und zu kurze Suchbegriffe (etwa »HUBR« oder »HUBAER« statt »HUBER«) erkennt. Da sie vollkommen in Maschinensprache geschrieben wurde, arbeitet sie recht schnell. Zudem ist sie nur wenige Bytes (drei Blocks) lang.

Laden Sie die Routine mit dem Befehl LOAD "GENIE-SUCH",8,8 von Diskette in den Speicher. Geschieht dies im Direktmodus, sollte danach NEW eingegeben werden, um alle Zeiger wieder richtigzustellen. Die Routine kann und soll aber auch von Basicprogrammen aus nachgeladen werden, etwa so:

```
1 IFA=0THENA=1:LOAD"GENIE-SUCH",8,8
```

Jetzt kann das Utility mit dem Befehl

```
SYS 49152,SU$,MODE,FE$(ANF),FE$(END),FI%
```

gestartet werden. Dabei haben die Parameter folgende Bedeutung: SU$ ist der nicht leere String, der den Suchtext enthält. Es kann sich um eine Variable, einen Text in Anführungszeichen oder einen Stringausdruck handeln. MODE gibt die Betriebsart an, eine Zahl oder Variable oder ein Term mit einem Wert zwischen 0 und 255 nach untenstehender Tabelle. FE$(...) ist die Arrayvariable, in der gesucht werden soll. Es soll sich um ein

eindimensionales Stringfeld handeln. ANF ist die erste Indexnummer, END die letzte Indexnummer. Diese beiden Parameter können auch größer als 255 sein, sie geben den Bereich im Array an, der durchsucht werden soll. ANF sollte nicht Null sein, da die Routine die Nummer des Feldes zurückmeldet, in der der Begriff gefunden wurde. Wird eine Null zurückgegeben, bedeutet dies, daß nichts gefunden wurde. Das Unterprogramm gibt aber auch dann eine Null zurück, wenn der Begriff im Feld Nummer Null gefunden wurde. Da in diesem Fall das steuernde Basicprogramm beide Fälle nicht unterscheiden kann, sollte die Null als Parameter hier vermieden werden. END darf nicht kleiner als ANF und nicht größer als die Anzahl der Elemente in FE$(...) sein. FI% schließlich muß eine Integer-Variable sein, in der die Nummer des Feldes zurückgemeldet wird, in dem der Suchbegriff gefunden wurde. Wird FI% Null, wurde nichts gefunden.

Für MODE gibt es 256 Möglichkeiten, die durch Addition der folgenden Optionen entstehen:

1 = Suche falsche Zeichen (»HABER« statt »HUBER«)
2 = Keine Berücksichtigung Groß/Kleinbuchstaben
4 = »Dreher« finden (»HUBRE« statt »HUBER«)
8 = Das Fragezeichen ist als Joker zugelassen
16 = Das erste Zeichen MUSS übereinstimmen
32 = Leerzeichen ignorieren
64 = überflüssige Zeichen ignorieren (»HUBBER« statt »HUBER«)
128 = fehlende Zeichen ignorieren (»HUER« statt »HUBER«)

Diese Werte können direkt für MODE eingesetzt werden. Ist MODE=0, wird nur dann ein Fund gemeldet, wenn der Suchbegriff wirklich 1:1 mit dem Feld übereinstimmt (alle Optionen abgeschaltet). Soll zum Beispiel die Suche nach falschen Zeichen (Option 1) und die Suche nach Drehern (Option 4) erlaubt sein, so addieren Sie 1 und 4 und setzen somit die 5 als MODE ein. Sind alle Optionen eingeschaltet, wird MODE=255. Beachten Sie jedoch, daß die Routine intern immer nur maximal eine Option pro Feld zuläßt. Ist der Suchbegriff etwa »PEULAENR«, und enthält ein Feld den Inhalt »PAULANER« (Fehler: ein falsches Zeichen und ein Dreher), so wird dieses Feld nicht gefunden, auch wenn Sie die beiden Optionen gleichzeitig erlaubt haben. Diese Einschränkung ist programmbedingt und mußte getroffen werden, um die Routine nicht unnötig komplex, umständlich und damit langsam werden zu lassen. Ausnahme: Die Optionen Leerzeichen ignorieren, Groß/Kleinschrift ignorieren, Fragezeichen als Joker können zusammen mit anderen Optionen verwendet werden.

Erklärung der Option 8: Das Fragezeichen kann als »Joker« verwendet werden, wenn Sie einzelne Zeichen des Suchbegriffes nicht wissen. Die Anwendung entspricht der des Fragezeichens beim Befehlskanal der Floppy: Geben Sie als Suchbegriff »PE?ER« vor, findet der Computer ein Feld mit dem Inhalt »PETER«, allerdings nur, wenn Sie die Option 8 zulassen.

Ein Anwendungsbeispiel: Das Feld A$(1) bis A$(15) enthält einige Vornamen. Sie wollen prüfen, in welchem Feld der »PETER« enthalten ist. Als Optionen lassen Sie zu: falsche Zeichen, Dreher, Vertauschung Groß/Kleinschrift, Leerzeichen ignorieren, Fragezeichen als Joker. Die Addition der Optionen ergibt MODE=47. Das Ergebnis soll in F% vermerkt werden. Der Befehl lautet:

```
SYS 49152,"PETER",47,A$(1),A$(15),F%
```

Wird nach diesem Befehl F%=5, so ist »PETER« in A$(5) enthalten. Ist F%=0, so ist »PETER« in A$(1) bis A$(15) nicht enthalten (aber vielleicht zum Beispiel in A$(16)).

Zur praktischen Anwendung: Eine sinnvolle Vorgehensweise wird sein, die Suchroutine erst einmal ohne alle Optionen durchlaufen zu lassen, um bei korrekter Eingabe auch wirklich nur das gewünschte Element zu finden. Blieb die Suche erfolglos, können Sie nun nacheinander einige SYS 49152-Aufrufe verwenden und dabei nacheinander die zur Verfügung stehenden Optionen einschalten.

Wir wünschen Ihnen viel Spaß mit dieser Routine, die Ihnen sicher helfen wird, Ihre Datenverwaltungsprogramme und andere Projekte komfortabler zu gestalten. Das Demoprogramm gibt noch ein weiteres Anwendungsbeispiel.

KAPITEL 2: ASSEMBLERPROGRAMMIERUNG

Dateibearbeitung in Maschinensprache

Man braucht keinen Generalschlüssel, um in Assembler Files zu öffnen, zu schließen, zu lesen und zu beschreiben. Es sind nur ein paar kleine Programmiertricks, die im Prinzip ganz ähnlich wie in Basic funktionieren. Unser kleiner Kurs für fortgeschrittene Assemblerprogrammierer führt Sie in die Dateibearbeitung direkt vom Prozessor aus.

Sie haben gerade Ihre ersten Schritte in der Welt der Maschinensprache hinter sich, wissen, wie man in Assembler Programme eingibt und startet, und sind nun daran interessiert, in größeren Programmen auch Arbeiten wie das Laden und Speichern von Dateien, Anzeigen des Directories, Fehlerkanal auslesen und so weiter direkt in Assembler zu programmieren. Dann sind Sie hier genau richtig. Ausgehend davon, wie man solche Dinge in Basic programmiert, werden wir uns mit Hilfe der Betriebssystem-Routinen langsam an Files in Maschinensprache herantasten. In bewährter Manier stellen wir dabei erst die Programmlistings vor, und kommentieren sie dann ausführlich. Den Abschluß bietet ein Programm, das den Inhalt einer beliebigen Datei auf dem Drucker wiedergibt. Übrigens funktionieren die hier vorgestellten Verfahren keineswegs nur auf dem C 64. Sie sind, soweit nur Kernal-Routinen mit der Startadresse $FFxx verwendet werden, ohne Änderung auf allen Commodore-Homecomputern lauffähig, insbesondere C 128, VC 20, C 16 und möglicherweise sogar auf dem Amiga. Allerdings kann es notwendig sein, die Startadresse der Beispielprogramm anzupassen. In diesem Artikel wurde einheitlich 49152 gewählt, da sich dieser Speicherbereich beim C 64 anbietet. Ohne Änderungen sind alle hier vorgestellten Programmlösungen auf dem C 64 lauffähig.

Öffnen und Schließen von Files

Wie Sie das von Basic gewohnt sind, müssen wir auch in Maschinensprache dem Computer genau mitteilen, auf welche Datei zugegriffen werden soll. Als Beispiel wählen wir eine auf Diskette gespeicherte sequentielle Datei mit dem Namen »TEST«. In Basic können wir so ein File ganz einfach zum Lesen öffnen:

```
OPEN 1,8,2,"TEST,S,R"
```

Die erste Zahl, 1, gibt die logische Filenummer an. Wir brauchen sie, damit wir uns später auf diesen OPEN-Befehl beziehen können. Die 8 gibt die Gerätenummer an, sie steht für das Diskettenlaufwerk. 2 ist eine Sekundäradresse, die der Floppy die Betriebsdaten mitteilt. Dahinter steht in Stringform erst der Dateiname, dann durch Kommas getrennt ein S für SEQ-Datei und ein R für READ (Lesen aus dem File).

Ganz so einfach geht's in Maschinensprache nicht. Den OPEN-Befehl müssen wir in drei Prozeduren zerlegen. Dabei begegnen uns allerdings exakt die selben Angaben wie bei Basic. Zunächst brauchen wir drei wichtige Betriebssystem-Routinen:

```
SETPAR  $FFBA
SETNAM  $FFBD
OPEN    $FFC0
```

Die SETPAR-Routine, die im System bereits fest ab Adresse $FFBA gespeichert ist, definiert die File- und die Gerätenummer sowie die Sekundäradresse. Diese Angaben sind im Akku, X- und Y-Register zu übergeben. Der Filename muß im Ascii-Code irgendwo im Speicher abgelegt werden. Dazu übergeben wir der SETNAM-Routine im Akku die Länge des Namens und in den Registern X und Y einen Zeiger (Low/Highbyte) auf die Adresse. Soll kein Dateiname gesetzt werden (ist ja auch in Basic bei OPEN nicht notwendig), übergeben wir als »Länge« im Akku die Null. Erst nach diesen Vorbereitungen darf die OPEN-Routine aufgerufen werden, die das File tatsächlich öffnet. Formulieren wir einmal den obigen OPEN-Befehl in Assembler. Die Funktionsweise der kurzen Routine ist dabei eher unwichtig, begreifen Sie den Ausschnitt einfach als »Kochrezept«:

```
  * = 49152 ; Startadresse
  LDA #1    ; Filenummer
  LDX #8    ; Gerätenummer
  LDY #2    ; Sekundäradresse
  JSR $FFBA ; SETPAR
  LDA #8    ; Länge des Filenamens
  LDX #<NAME; Adresse
  LDY #>NAME; des Filenamens
  JSR $FFBD ; SETNAM
  JSR $FFC0 ; OPEN Datei öffnen
```

Es ist also ganz einfach. Diese wenigen Zeilen haben exakt die gleiche Wirkung wie der oben vorgestellte OPEN-Befehl. Nur eines fehlt noch: Der Dateiname, den wir an einer freien Stelle im Speicher im Ascii-Code ablegen müssen:

```
NAME   .ASC "TEST,S,R"
```

Das zweite, was jeder Basicprogrammierer im Zusammenhang mit Files lernt, ist, daß man jede geöffnete Datei wieder schließen muß, wenn man sie nicht mehr braucht. Das ist so, als ob Sie Ihren Computer einschalten (OPEN), dann damit arbeiten und ihn nach Beendigung der Session wieder abschalten (CLOSE). Nur, daß Sie mit mehreren Computern gleichzeitig arbeiten können: Es können viele Dateien gleichzeitig geöffnet sein. Auf welche Sie sich momentan beziehen, wird durch die Filenummer (in unserem Beispiel 1) eindeutig bestimmt. In Basic schließen wir die oben geöffnete Datei einfach mit

CLOSE 1

Das geht in Maschinensprache fast genauso einfach: Die entsprechende CLOSE-Routine erwartet im Akku die Filenummer und hat die Adresse $FFC3. Unser CLOSE 1 sieht also in Assembler so aus:

```
  LDA #1    ; Dateinummer
  JSR $FFC3 ; CLOSE Datei schließen
```

Das ist wirklich alles! Bemerkenswert ist, daß wir hier ebenso wie in Basic keine weiteren Angaben brauchen - die hat ja bereits alle der OPEN-Befehl bekommen.

Lesen aus der Datei

Nachdem wir das File geöffnet haben, wollen wir nun daraus lesen. In Assembler muß dazu zunächst ein Eingabekanal geöffnet werden, das heißt, wir müssen dem Computer sagen, daß wir die Datei, die vorher geöffnet wurde, nun lesen wollen. Dazu gibt es die CHKIN-Routine mit der Adresse $FFC6. Diesem Unterprogramm wird im X-Register (!) die Filenummer der vorher geöffneten Datei übergeben, aus der gelesen werden soll. In unserem Beispiel sieht das so aus:

```
  LDX #1    ; Dateinummer 1
  JSR $FFC6 ; CHKIN Eingabekanal öffnen
```

Ab jetzt kann man aus dieser Datei lesen. Dazu stehen im Prinzip vor allem zwei Routinen zur Verfügung:

```
GETIN   $FFE4
CHRIN   $FFCF
```

Die CHRIN-Routine entspricht in etwa dem INPUT# in Basic. Sie merken schon: In Basic muß hinter INPUT# noch die Filenummer angegeben werden. Das kann hier entfallen, da wir ja schon mit CHKIN festgelegt haben, auf welches File Bezug genommen wird. Die GETIN-Routine entspricht dem GET#-Befehl von Basic und ist CHRIN grundsätzlich vorzuziehen. Bitte begnügen Sie sich mit der Ausrede, GET in Basic ist ja auch sicherer als INPUT. Eine etwas detailliertere Erklärung folgt am Ende des Artikels. Die Anwendung von GETIN ist denkbar einfach: Einfach mit JSR $FFE4 aufrufen. Im Akku wird dann der Ascii-Code des gelesenen Zeichens übergeben. Auf ein Beispiel verzichten wir, da in Kürze das erste »richtige« Programm folgt.

Ebenso wie geöffnete Dateien wieder geschlossen werden, müssen auch die Eingabekanäle (später werden wir noch Ausgabekanäle kennenlernen) geschlossen werden, da sonst beispielsweise keine Eingabe von Tastatur mehr möglich ist. Unser C 64 kennt dazu die Routine CLRCHN mit der Adresse $FFCC. Rufen Sie diese Adresse auf, wird die Eingabe wieder auf die Tastatur und (falls verändert) die Ausgabe wieder auf den Bildschirm »umgelenkt« (man sollte besser sagen: »zurückgelenkt«).

Zur Sicherheit sollte man außerdem zu Beginn jedes Maschinenprogramms den CLALL-Befehl (Adresse $FFE7) anwenden. Er bewirkt, daß alle offnene Dateien und Kanäle geschlossen werden. CLALL beinhaltet also CLRCHN.

Der Vollständigkeit halber sei noch die wohl wichtigste Betriebssystem-Routine erwähnt, die es gibt: CHROUT (oft auch PRINT oder

BSOUT genannt) mit der Adresse $FFD2. Diese gibt einfach das Zeichen, dessen Codenummer im Akku steht, auf dem Bildschirm oder dem Gerät aus, das momentan Ausgabegerät ist. Wie man ein anderes Ausgabegerät als den Schirm definiert (funktioniert ähnlich wie oben bei CHKIN gesehen), lernen Sie später.

Fehlerkanal auslesen

Mit dem bisher errungenen Wissen können wir schon unser erstes Programm schreiben. Es soll den Fehlerkanal der Diskettenstation auslesen und am Bildschirm anzeigen. Dazu erst das entsprechende Analogon in Basic:

```
10 OPEN 15,8,15:REM KEIN DATEINAME
20 GET# 15,A$
30 PRINT A$;
40 IF A$ <> CHR$(13) THEN 20
50 CLOSE 15
60 END
```

In Zeile 40 erfolgt eine Prüfung, ob das gelesene Zeichen ein CR (Code 13) war. In diesem Fall ist die Statusmeldung vollständig gelesen. Dieses Programm setzen wir 1:1 in Maschinensprache um.

```
       ; Beispiel 1: Fehlerkanal
49152  JSR $FFE7; CLALL zur Sicherheit
49155  LDA #15  ; Filenummer
49157  LDX #8   ; Floppy
49159  TAY      ; Sekundäradresse
49160  JSR $FFBA; SETPAR
49163  LDA #0   ; kein Dateiname
49165  JSR $FFBD; SETNAM
49168  JSR $FFC0; OPEN 15,8,15,""
49171  LDX #15  ; File 15
49173  JSR $FFC6; CHKIN zur Eingabe öffnen
49176  JSR $FFE4; GETIN ein Zeichen lesen
49179  JSR $FFD2; auf Bildschirm ausgeben
49182  CMP #13  ; Code 13?
49184  BNE 49176; nein, dann weiter
49186  JSR $FFCC; sonst Kanal schließen
49189  LDA #15  ; Dateinummer
49191  JMP $FFC3; CLOSE 15
```

Das war auch schon alles. Wenn Sie dieses Programm mit dem Monitor oder Assembler eingegeben und ggf. assembliert haben, starten Sie es mit

```
SYS 49152
```

Sofort erscheint der Fehlerkanal, im Normalfall 00,OK,00,00. Senden Sie einfach einmal einen fehlerhaften Befehl, etwa

```
OPEN 1,8,15,"X":CLOSE 1
```

Nach SYS 49152 beschwert sich das Laufwerk mit einem 31,SYNTAX ERROR,00,00. Was bleibt ihm auch anderes übrig: Den X-Befehl gibt es nicht. Übrigens sollten Sie zur Probe im Maschinenprogramm einmal an Adresse 49189 ein RTS einbauen (POKE 49189,96) und damit den CLOSE-Befehl unwirksam machen. Der Aufruf mit SYS 49152 klappt einwandfrei, allerdings ist die Datei 15 noch offen. Das können Sie leicht nachprüfen, indem Sie jetzt von Basic aus

eingeben

OPEN 15,8,15

Der Computer reagiert artig mit seinem ?FILE OPEN ERROR. In Maschinensprache haben wir es also offenbar mit exakt den gleichen Files zu tun wie in Basic. Das erleichtert die Sache erheblich.

Directory ohne Datenverlust

Kaum ein größeres Programm kommt ohne eine Routine aus, die das Inhaltsverzeichnis der Diskette lädt und listet. In Basic bietet sich dazu folgende Befehlsfolge an:

```
LOAD "$",8
LIST
```

die jedoch einen fatalen Nebeneffekt hat: Da das Directory wie ein Basicprogramm geladen wird, geht das momentan im Speicher stehende Programm rettungslos verloren. Versuchen wir daher, eine kurze Maschinenroutine zu programmierern, die das gleiche ohne Datenverlust erledigt. Wir lesen Zeichen für Zeichen der Directory und zeigen alles gleich an. Die hier vorgestellte Routine ist nicht optimiert in Bezug auf Kürze und Geschwindigkeit, aber es wird Ihnen mit etwas Einfühlungsvermögen leichtfallen, die Funktionsweise zu verstehen und das Programm vielleicht noch zu erweitern oder zu kürzen.

Zunächst öffnen wir in gewohnter Manier das »File«, in dem sich die Directory verbirgt. Aus technischen Gründen braucht man dazu die Sekundäradresse 0.

```
       ; Beispiel 2: Directory
49152  JSR $FFE7; CLALL zur Sicherheit
49155  LDA #1   ; Filenummer
49157  LDX #8   ; Geräteadresse
49159  LDY #0   ; Sekundäradresse
49161  JSR $FFBA; SETPAR
49164  LDX #$24 ; Code für Dollarzeichen
49166  STX 2    ; in freier Speicherzelle speichern
49168  LDX #2   ; Adresse low
49170  LDY #0   ; und high
49172  JSR $FFBD; SETNAM
49175  JSR $FFC0; OPEN 1,8,0,"$"
49178  LDX #1   ; Filenummer
49180  JSR $FFC6; CHKIN Eingabekanal öffnen
```

Da ja der Filename irgendwo im Speicher abgelegt werden muß, entscheiden wir uns für die sonst unbenutzte Speicherzelle 2. Ab 49168 richten wir den Zeiger X/Y auf diese Adresse. Sie wundern sich vielleicht, warum wir vor SETPAR nicht die Länge des Namens (1) im Akku gespeichert haben. Das ist hier nicht mehr nötig, da die 1 im Akkumulator von 49155 noch enthalten ist. Ein kleiner Trick also, den sich der Autor einfach nicht verkneifen konnte. Jetzt können wir mit GETIN Zeichen für Zeichen lesen und ausgeben. Beim Directory gibt es allerdings einige Besonderheiten zu beachten. Die ersten beiden Bytes einer jeden Zeile können ignoriert werden, die nächsten beiden Bytes geben im Format Low/Highbyte die Größe in Blocks an. Danach folgt bis zum Nullbyte die Zeileninformation. Außerdem muß getestet werden, ob das Dateiende erreicht wurde. Wie in Basic lesen wir dazu den Wert der Variablen

ST aus, auf den der Maschinenprogrammierer durch Auslesen der Speicherzelle 144 (C 64, C 128 und andere) Zugriff hat. Hat diese Speicherzelle einen Inhalt ungleich Null, so ist ein Fehler aufgetreten oder das File ist beendet. Zuletzt brauchen wir eine Routine, die eine 16-Bit Integerzahl (Low/Highbyte) numerisch auf dem Bildschirm ausgibt. So etwas enthält der C 64 an Adresse $BDCD, diese Routine AXOUT wird sonst vom System zur Ausgabe von Basic-Programmzeilennummern verwendet. Sie gibt eine Zahl aus, die sich nach der Formel

WERT = X + A * 256

errechnet. High- und Lowbyte wird also im Akku und im X-Register übergeben.

```
49183  LDY #3   ; drei Bytes überlesen
49185  STY 3    ; als Zähler merken
49187  JSR $FFE4; GETIN Zeichen lesen
49190  STA 4    ; und merken
49192  LDY 144  ; ST Status lesen
49194  BNE 49239; ungleich Null, fertig
49196  JSR $FFE4; GETIN
49199  LDY 144  ; ST lesen
49201  BNE 49239; bei Dateiende
49203  LDY 3    ; Zähler zurückholen
49205  DEY      ; und erniedrigen
49206  BNE 49185; nicht null, weitermachen
49208  LDX 4    ; Zeichen restaurieren
49210  JSR $BDCD; AXOUT Zahl ausgeben
49213  JSR $AB3F; SPACE Leerzeichen ausgeben
49216  JSR $FFE4; GETIN
49219  LDX 144  ; Status testen
49221  BNE 49239; bei ungleich Null beenden
49223  TAX      ; Zeichencode
49224  BEQ 49232; bei Zeilenende
49226  JSR $FFD2; BSOUT Zeichen ausgeben
49229  JMP 49216; und weiter listen
49232  JSR $AAD7; CRLF Zeilenende ausgeben
49235  LDY #2   ; zwei Bytes für Linkpointer
49237  BNE 49185; und nächste Zeile
49239  JSR $FFCC; CLRCHN Kanal schließen
49242  LDA #1   ; Datei 1
49244  JMP $FFC3; schließen, fertig
```

Besitzer anderer Computer als der C 64 müssen die Befehle JSR $AAD7 (beginnt neue Zeile) und JSR $AB3F (gibt ein Leerzeichen aus) evtl. durch Ersatzkonstruktionen ersetzen. JSR $AB3F hat die gleiche Wirkung wie LDA #32 und JSR $FFD2. Außerdem müssen Sie mit Hilfe eines ROM-Listings die AXOUT-Routine, die beim C 64 an $BDCD beginnt suchen, oder auf die Ausgabe der Blocklänge verzichten.

Probieren Sie dieses Programm gleich einmal aus! Wenn alles geklappt hat, sollten Sie zur Übung einmal versuchen, beispielsweise eine Funktion einzubauen, die das Directory auf Tastendruck anhält und/oder abbricht.

Druckerausgabe

Nachdem wir gelernt haben, wie man in Maschinensprache Eingaben von Files vornimmt, beschäftigen wir uns jetzt mit der Ausgabe in eine Datei. Als erstes Beispiel mag eine Druckerausgabe dienen. Wenn Sie das bisher

beschriebene verstanden haben, wird es Ihnen nicht schwerfallen, auch mit dem folgenden zurechtzukommen. Im Prinzip geht es wieder wie gehabt, nur daß wir diesmal nach dem Öffnen des eigentlichen Files keinen Eingabekanal, sondern einen Ausgabekanal öffnen werden. Dazu dient die Routine CHKOUT (Adresse $FFC9), die analog zu CHKIN funktioniert. Auch hier muß im X-Register die Filenummer übergeben werden. CHKOUT leitet dann bis zum abschließenden CLRCHN alle Ausgaben auf das betreffende Gerät um. Das funktioniert genauso wie der CMD-Befehl, den Sie vielleicht vom Basic her kennen. Und in der Tat ruft auch Basics CMD intern einfach nur die CHKOUT-Routine auf!

Es soll der Text »ICH GRUESSE DIE WELT!« auf dem Drucker ausgegeben werden. Sehen wir uns wieder zunächst die Basic-Lösung an:

```
10 OPEN 4,4
30 PRINT#4,"ICH GRUESSE DIE WELT!"
50 CLOSE 4
```

Die Zeilen 20 und 40 wurden bewußt weggelassen. Damit die Umsetzung in Assembler leichter zu verstehen ist, bauen wir das Basicprogramm etwas um und verwenden den CMD-Befehl. Dieser ergibt die noch fehlende Zeile 20. Bekanntlich muß ein CMD-Befehl mit einem PRINT#-Befehl aufgehoben werden, den wir in Zeile 40 unterbringen.

```
10 OPEN 4,4
20 CMD 4
30 PRINT"ICH GRUESSE DIE WELT!"
40 PRINT#4
50 CLOSE 4
```

Das Bemerkenswerte dabei ist, daß wir zur Druckerausgabe den normalen PRINT-Befehl (ohne Doppelkreuz) verwenden dürfen, da ja zuvor mit CMD schon der entsprechende Kanal freigegeben wurde. In Maschinensprache können wir uns die Zeile 40 sparen, wir haben zum Schließen des Kanals ja die CLRCHN-Routine zur Verfügung, die übrigens auch dem Basic-Programmierer mit SYS 65484 (hier etwa in Zeile 40) zur Verfügung steht.

Jetzt stellt die Umsetzung kein Problem mehr dar. Merken Sie sich bitte, daß beim Öffnen eines Druckerfiles grundsätzlich kein Filename angegeben wird. Eventuell kann es notwendig sein, eine andere Sekundäradresse als die Null anzugeben, das hängt von Ihrem Interface ab. Ändern Sie bei Bedarf einfach den Befehl ab Adresse 49158.

```
       ; Beispiel 3: Druckerausgabe
49152  JSR $FFE7; CLALL zur Sicherheit
49155  LDA #4   ; Filenummer
49157  TAX      ; istgleich Gerätenummer
49158  LDY #0   ; Sekundäradresse
49160  JSR $FFBA; SETPAR
49163  LDA #0   ; Länge des Filenamens = 0
49165  JSR $FFBD; SETNAM
49168  JSR $FFC0; OPEN 4,4,0,""
49171  LDX #4   ; Filenummer
49173  JSR $FFC9; CHKOUT Drucker als Ausgabegerät
```

Jetzt ist die eigentliche Ausgabe des Textes dran. Wir verwenden dazu, damit es leichter verständlich wird, eine Schleife, in der wir Zeichen für Zeichen des im Ascii-Code abgelegten Textes über CHROUT ($FFD2) ausgeben. Im

Gegensatz zu Basic wird hier bei der Ausgabe nicht automatisch das Zeilenendezeichen CR (CHR$(13)) angefügt, das müssen wir von Hand erledigen. Wird das vergessen, druckt das Gerät nichts. Der Text hat insgesamt also 22 Zeichen.

```
49176  LDX #0      ; Zähler initialisieren
49178  LDA 49197,X ; Text lesen
49181  JSR $FFD2   ; und Zeichen ausgeben
49184  INX         ; nächstes Zeichen
49185  CPX #22     ; schon 22 Zeichen?
49187  BNE 49178   ; nein, weiter lesen
49189  JSR $FFCC   ; CLRCHN Ausgabe beenden
49192  LDA #4      ; File 4
49194  JMP $FFC3   ; CLOSE 4
49197  .ASC "ICH GRUESSE DIE WELT!"
49218  .BYT 13     ; CR
```

Je nachdem, welchen Assembler oder Monitor Sie verwenden, kann es notwendig sein, die Pseudo-Befehle in 49197 und 49218 zu ändern. Es soll einfach der Text ab 49197 im Ascii-Code abgelegt und eine 13 in Speicherzelle 49218 gelegt werden: POKE 49218,13.

Jetzt sollten Sie den Drucker einschalten und bereitmachen. Mit

SYS 49152

kann dann der Ausdruck gestartet werden. Wenn alles richtig gelaufen ist, wird das Gerät schön brav den Text zu Papier bringen.

Beschreiben eines Disketten-Files

Jetzt wollen wir aber auch versuchen, in ein File zu schreiben. Dazu werden wir eine neue Routine des C 64 kennenlernen, die uns die Arbeit des Aufrufs von SETPAR und SETNAM abnimmt. Man kann vielmehr direkt hinter dem SYS-Befehl Filenamen und Gerätenummer angeben. Diese Routine GETPAR (Adresse beim C 64: $E1D4) wird normalerweise von den LOAD- und SAVE-Befehlen verwendet, um die Parameter zu ermitteln. Wir wollen ein kurzes Programm schreiben, mit dem eine sequentielle Datei angelegt werden kann.

Kurz zu den »Spielregeln«: Das Programm wird mit

SYS 49152,"DATEINAME,S,W",8

gestartet. Daraufhin erscheint ein Cursor, und die einzelnen Datensätze werden durch Komma getrennt eingegeben. Das Ende markiert ein Stern, der sonst nicht eingegeben werden darf - der Einfachheit halber. Doch wie programmiert man so etwas?

Zunächst holen wir das Komma hinter SYS 49152 und die Parameter. Dabei müssen die Sekundäradresse und die Filenummer immer auf 2 gesetzt werden, auch wenn der Anwender nichts oder etwas anderes vorgibt.

```
       ; Beispiel 4: File-Editor
49152  JSR $AEFD; CHKKOM Komma holen
49155  JSR $E1D4; GETPAR Parameter holen
49158  LDX $BA  ; Gerätenummer G aus Vorgabe
49160  LDA #2   ; Filenummer
49162  TAY      ; istgleich Sekundäradresse
```

```
49163  JSR $FFBA; SETPAR
49166  JSR $FFC0; OPEN 2,G,2,"NAME"
```

Zur Eingabe von Tastatur verwenden wir diesmal die oben geschmähte CHRIN-Routine. Wendet man diese auf die Tastatur an, wird beim ersten Aufruf ein Cursor ausgegeben, und der Anwender kann etwas eingeben und mit <RETURN> bestätigen. Im Akku steht dann das erste Textzeichen. Bei weiteren Aufrufen werden dann nacheinander die einzelnen Zeichen gelesen. Das Ganze ist von der Wirkung her mit dem INPUT-Befehl vergleichbar.

```
49169  LDX #2   ; Filenummer
49171  JSR $FFC9; CHKOUT Ausgabekanal in File 2 öffnen
49174  JSR $FFCF; CHRIN Eingabe holen
49177  CMP #42  ; Stern?
49179  BEQ 49187; ja, dann fertig
49181  JSR $FFD2; sonst in File ausgeben
49184  JMP 49174; und weitermachen
49187  JSR $FFCC; CLRCHN Kanal schließen
49190  LDA #2   ; Filenummer
49192  JMP $FFC3; CLOSE 2
```

Beim Schreiben in eine Datei auf Diskette ist es ganz wichtig, daß nach dem Beschreiben ordentlich der Kanal mit CLRCHN und das File mit CLOSE geschlossen werden, da sonst die Datei nicht ordnungsgemäß angelegt wird. Wenden wir das Programm einmal an. Legen Sie eine nicht schreibgeschützte Diskette ein, auf der sich noch Platz befindet, und starten das Programm mit

```
SYS 49152,"TEST,S,W",8
```

Wählen Sie einen Namen, den es noch nicht auf dieser Diskette gibt! Das Laufwerk beginnt kurz zu arbeiten, und ein Cursor erscheint. Geben Sie nun einige Zeilen Text ein, bestätigen Sie wie in Basic jede Zeile mit <RETURN>. Wenn Sie genug haben, geben Sie nur einen Stern gefolgt von <RETURN> ein. Die Datei wird nun geschlossen, der Computer kehrt ins Basic zurück. Wenn Sie nun das Directory einsehen, stellen Sie fest, daß eine sequentielle Datei »TEST« erzeugt wurde. Wir haben also einen richtigen kleinen Editor für sequentielle Files geschrieben! Versuchen Sie doch einmal, das Programm so auszubauen, daß auch Änderungen an bereits bestehenden Files vorgenommen werden können. Auch erkennt unsere Routine keine Diskettenfehler, man müßte dazu noch den Fehlerkanal auslesen, beispielsweise nach dem Schließen der Datei, und prüfen, ob eine andere Angabe als 00,OK,00,00 ausgegeben wurde. Es reicht in diesem Fall, nur das erste Zeichen des Fehlerkanals zu lesen und mit der Null zu vergleichen.

File-Lister

Den krönenden Abschluß stellt ein kleines Hilfsprogramm dar, mit dessen Hilfe eine Datei von Diskette gelesen werden und direkt auf den Drucker ausgegeben werden kann. Die Bedienung soll dabei wieder über Angaben nach dem SYS-Befehl erfolgen, also listet beispielsweise

```
SYS 49152,"TEST,S,R"
```

die eben erzeugte sequentielle Datei auf dem Drucker. Im Prinzip ist das gar nicht schwer. Wir beginnen damit, ein Diskettefile zu öffnen und dann einen Ausgabekanal zum Drucker anzulegen.

```
       ; Beispiel 5: File-Lister
```

```
49152  JSR $AEFD; CHKKOM Komma holen
49155  JSR $E1D4; GETPAR Parameter holen
49158  LDX $BA  ; Gerätenummer G aus Vorgabe
49160  LDA #2   ; Filenummer
49162  TAY      ; istgleich Sekundäradresse
49163  JSR $FFBA; SETPAR
49166  JSR $FFC0; OPEN 2,G,2,"NAME"
49169  LDA #0   ; kein Filename
49171  JSR $FFBD; SETNAM
49174  LDA #4   ; Filenummer
49176  TAX      ; istgleich Gerätenummer Drucker
49177  LDY #0   ; Sekundäradresse
49179  JSR $FFBA; SETPAR
49182  JSR $FFC0; OPEN 4,4,0
```

Zwei Dinge haben wir dabei gelernt. Erstens spielt die Reihenfolge des Aufrufs von SETNAM und SETPAR keine Rolle. Beim Drucker haben wir erst die Angaben zum Filenamen und dann die Angaben zu den Fileparametern gemacht, anders als gewohnt. Wichtig ist nur, daß beide Routinen vor dem OPEN bereits aufgerufen wurden. Zweitens ist es wie auch in Basic problemlos möglich, mehrere Files auf einmal zu öffnen. Diese müssen sich lediglich in der Filenummer unterscheiden, damit wir uns später auf sie beziehen können.

Jetzt schalten wir File 2 (SEQ-Datei) auf einen Eingabekanal und holen uns ein Zeichen.

```
49185  LDX #2   ; Filenummer
49187  JSR $FFC6; CHKIN
49190  JSR $FFE4; GETIN
```

Planen wir im Voraus. Später wird unser Programm prüfen müssen, ob das Dateiende erreicht ist. Wie checken den Inhalt des Status. Wie das geht, wissen Sie ja schon: Speicherzelle 144 auslesen. Das Programm kann damit allerdings nicht zu lang warten, da sich der Status während späterer Operationen, beispielsweise bei der Zeichenausgabe zum Drucker noch ändern kann. Also lesen und speichern wir ST jetzt:

```
49193  LDX 144  ; Statusbyte
49195  PHP      ; auf den Stack
```

Wir verwenden das X-Register, da sich im Akku ja noch das auszugebende Zeichen von GETIN befindet. Der LDX-Befehl setzt das Zero-Flag des Prozessors, wenn eine Null gelesen wurde (Dateiende nicht erreicht). PHP sichert die Prozessorflags zur späteren Untersuchung auf den Stack (PHush Processorstatus), bis wir sie brauchen.

Bevor wir jetzt allerdings das Zeichen drucken können, müssen wir den Eingabekanal mit CLRCHN wieder schließen. Fragen Sie nicht, es scheint bei Commodore-Computern einfach notwendig zu sein. Dabei müssen wir allerdings vorsichtig sein: Der CLRCHN-Aufruf ändert den Inhalt des Akkumulators, das Zeichen geht verloren. Deshalb pushen wir es auf den Stack.

```
49196  PHA      ; Zeichen retten
49197  JSR $FFCC; CLRCHN
```

Der Stack enthält nun zwei Dinge: Erstens das gerade gelesene Zeichen und zweitens den Prozessor-Status (nicht mit dem Statusbyte ST verwechseln! Der PHP-Befehl legt nicht den Inhalt des X-Registers auf den Stack, sondern

unter anderem eine Information darüber, ob der letzte Lesebefehl in 49193 Null ergab!). Wenn wir damit anfangen, Werte vom Stack zu holen, bekommen wir zuerst die Werte, die zuletzt dorthin geschrieben wurden - in unserem Fall das Zeichen.

Die Druckerausgabe enthält keine Geheimnisse, hier ist der Ausschnitt:

```
49200  LDX #4   ; Druckerfile
49202  JSR $FFC9; CHKOUT Ausgabekanal öffnen
49205  PLA      ; Zeichen vom Stack retten
49206  JSR $FFD2; und zum Drucker schicken
49209  JSR $FFCC; CLRCHN Ausgabekanal schließen
```

Gibt es noch gültige Daten, die wir aus dem Eingabekanal lesen müssen? Der folgende Code prüft das:

```
49212  PLP      ; Prozessorstatus zurückholen
49213  BEQ 49185; weitere Daten, dann oben weiter
```

Sind keine Zeichen mehr in der Datei, können wir beide Files schließen: Zum Beispiel erst das Druckerfile, dann die SEQ-Datei:

```
49215  LDA #4   ; Drucker
49217  JSR $FFC3; CLOSE 4
49220  LDA #2   ; Diskette
49222  JMP $FFC3; CLOSE 2 und fertig
```

Das war alles, was wir brauchen. Dieses kurze Utility können Sie nun allgemein verwenden. Um beispielsweise die oben angelegte Datei zu drucken, geben Sie den Befehl

```
SYS 49152,"TEST,S,R",8
```

Solche File-Lister haben in der Praxis eine große Bedeutung, wenn ohne Datenverlust Dateien ausgelesen werden sollen. Sie sind übrigens bei der Ausgabe nicht nur auf SEQ-Files beschränkt. Auch PRG- und vor allem USR-Files lassen sich listen, indem Sie die Kennung ‚S im Filenamen entsprechend in ‚P oder ‚U ändern. Vielleicht bauen Sie ja auch noch eine Art »Schalter« ein, mit dem man die Ausgabe wahlweise auf Bildschirm und/oder Drucker legen kann.

Laden einer Datei

Bisher haben wir noch nicht versucht, ein File an einen bestimmten Speicherplatz zu laden. Natürlich gibt es das Gegenstück zum LOAD-Befehl auch in Maschinensprache, und zwar in Form der LOAD-Routine mit der Adresse $FFD5. Auch hier müssen Sie erst mit SETPAR und SETNAM oder aber GETPAR Filenamen, Gerätenummer und Sekundäradresse festlegen, die logische Filenummer spielt keine Rolle. Der Inhalt des Akkumulators beim Aufruf der Routine entscheidet darüber, ob geladen wird (A=0) oder ein VERIFY ausgeführt (A ungleich 0). Ist die Sekundäradresse nicht Null, so wird das File »absolut« geladen, also an die Stelle, die auf Diskette gespeichert ist. Sie kennen diese Betriebsart, wenn Sie in Basic mit

```
LOAD "NAME",8,1
```

etwa Tools an eine vorgegebene Stelle im Speicher laden. Man kann aber auch »relativ« laden, dem Computer also sagen: »Ignoriere die auf Diskette

gespeicherte Ladeadresse und lade das File auf jeden Fall an Adresse xxx«. Diese Betriebsart findet beim Laden von Basicprogrammen Verwendung, da lassen Sie hinter LOAD die ,1 einfach weg. Für die Sekundäradresse wird Null gesetzt. In Basic wird dann immer an die Adresse geladen, die in den Speicherzellen 43/44 steht, normalerweise der Basic-Anfang 2049. In Assembler können wir der LOAD-Routine die Adresse mitgeben. Schreiben wir doch einmal ein Utility, mit dem man Hires-Grafiken unabhängig, ab wo sie gespeichert wurden immer nach 8192 laden kann. Dazu holen wir uns erst mit CHKKOM und GETPAR die Fileparameter. Dann setzen wir die Sekundäradresse (gespeichert in $B9) auf Null, um relativ zu laden. Die Ladeadresse 8192 (Highbyte: 32, Lowbyte: 0) übergeben wir im X- und Y-Register, im Akku eine Null, um zu laden, nicht zu vergleichen. Fertig sieht das so aus:

```
       ; Beispiel 6: Grafiklader
49152  JSR $AEFD; CHKKOM Komma holen
49155  JSR $E1D4; GETPAR Parameter holen
49158  LDA #0   ; LOAD-Flag
49160  TAX      ; Lowbyte von 8192
49161  LDY #32  ; Highbyte von 8192 ($2000)
49163  STA $B9  ; Sekundäradresse Null
49165  JMP $FFD5; LOAD, Grafik in den Speicher laden
```

Mehr ist es nicht! Jetzt könnten Sie beispielsweise mit

```
SYS 49152,"DATEINAME",8
```

ein Bild (PRG-File) nach 8192 laden. Soll die auf Diskette gespeicherte Adresse berücksichtigt werden, setzen Sie einfach die Speicherzelle $B9 (Sekundäradresse, dezimal 185) auf einen Wert ungleich Null. Die Übergabe einer Adresse in X und Y kann dann entfallen.

Speichern

Was nun noch fehlt, ist die Möglichkeit, einen beliebigen Speicherbereich auf Diskette zu speichern. In Basic speichert der SAVE-Befehl das Basicprogramm. Dessen Grenzen sind in den Speicherzellen-Paaren 43/44 (Anfang) und 45/46 (Ende) erfaßt. Wir wollen ein Grafikbild, das im Speicher von 8192 ($2000) bis 16383 ($3FFF) einschließlich steht, speichern. Vor dem Aufruf der SAVE-Routine ($FFD8) müssen Sie erst wieder mit SETPAR und SETNAM oder aber in unserem Fall GETPAR die Datei-Parameter festlegen. Dann wird im X- und Y-Register das High- und Lowbyte der um eins erhöhten Endadresse übergeben. Die Startadresse des zu speichernden Bereiches übergeben wir in einem beliebigen Speicherzellen-Paar der Zeropage (in unserem Fall etwa Adressen 2 und 3), dessen Adresse wiederum im Akku übergeben wird. Die Sekundäradresse ist beim Speichern ebenso wie die Filenummer ohne Bedeutung. Das alles klingt sehr umständlich, ist es aber gar nicht, wie das folgende Beispiel beweist:

```
       ; Beispiel 7: Grafik speichern
49152  JSR $AEFD; CHKKOM Komma holen
49155  JSR $E1D4; GETPAR
49158  LDX #0   ; bis $4000 ausschließlich speichern (Lowbyte 0)
49160  LDY #64  ; Highbyte von $4000
49162  STX 2    ; ab $2000 speichern (Lowbyte 0)
49164  LDA #32  ; Highbyte von $2000
49166  STA 3    ; merken
49168  LDA #2   ; Startadresse in Speicherzellen 2 und 3
49170  JMP $FFD8; SAVE-Routine: Bild speichern
```

Na sehen Sie, gar keine Schwierigkeit. Die SAVE-Routine erzeugt auf Diskette ein PRG-File mit in diesem Fall 33 Blocks Länge, das im normalen Format (erst zwei Bytes Ladeadresse, hier etwa 8192, dann die Datenbytes aus dem Speicher) das Grafikbild enthält.

Das war's!

Zum Abschluß noch ein Hinweis zu den Routinen SETPAR (in der Fachliteratur auch oft SETLFS genannt) und SETNAM. Wie Sie wissen, dienen die beiden Routinen dazu, vor Dateibefehlen die File-, Gerätenummer, Sekundäradresse sowie den Dateinamen zu definieren. Werfen wir einen Blick hinter die Kulissen: Hier je ein komplettes Assemblerlisting, wie SETPAR und SETNAM intern funktionieren:

```
**************************  SETPAR Fileparameter setzen
FFBA  4C 00 FE  JMP $FE00   SETPAR-Sprung
FE00  85 B8     STA $B8     logische Filenummer
FE02  86 BA     STX $BA     Geräteadresse
FE04  84 B9     STY $B9     Sekundäradresse
FE06  60        RTS         und fertig

**************************  SETNAM Filenamen setzen
FFBD  4C F9 FD  JMP $FDF9   SETNAM-Sprung
FDF9  85 B7     STA $B7     Länge
FDFB  86 BB     STX $BB     Adresse low
FDFD  84 BC     STY $BC     Adresse high
FDFF  60        RTS         und fertig
```

Die Routinen SETPAR und SETNAM schreiben also nur die Inhalte der Register A, X und Y in je drei Speicherzellen, es werden also im Prinzip nur je drei POKEs ausgeführt. Was lernen wir daraus? Richtig, es ist nicht immer optimal, JSR $FFBA oder $FFBD aufzurufen. Betrachten Sie das Beispiel 5 (File-Lister). Hier sollte ab Adresse 49158 eigentlich nur die File- und Gerätenummer auf 2 gesetzt werden, auch wenn der Anwender etwas anderes oder gar nichts angegeben hat. Da sich beim Aufruf von SETPAR in 49163 jedoch ein definierter Wert auch im X-Register (Gerätenummer) befinden muß, mußten wir umständlich in 49158 die alte Gerätenummer in das Register laden, nur damit sie dann von SETPAR in $FE02 wieder gepoket werden kann. Einfacher wäre es also hier (und übrigens auch in Beispiel 4) gewesen, wenn wir den Teil von 49158 bis 49163 ersetzt hätten durch

```
  LDA #2
  STA $B8  ; logische Filenummer
  STA $B9  ; Sekundäradresse
```

Dadurch hätten wir zwei Bytes gespart. Ebenso könnte man die Befehlsfolge

```
  LDA #0   ; kein Filename
  JSR $FFBD; SETNAM
```

die beim Eröffnen eines Druckerfiles die Länge des Filenamens auf Null setzt (vergleiche z.B. Beispiel 3 ab 49163) durch die beiden Befehle

```
  LDA #0   ; kein Filename
  STA $B7  ; Länge des Filenamens
```

ersetzen. Ersparnis: Ein Byte. Außerdem läuft die Alternative erheblich

schneller als die »Kochrezept-Version«.

Damit wären wir am Ende unseres kleinen Streifzugs durch die gar nicht so weite Welt der Ein/Ausgabe in Assembler angelangt. Experimentieren Sie doch noch ein wenig mit dem Gelernten, das kann sicher nicht schaden! Zum Abschluß noch eine tabellarische Übersicht über die behandelten Betriebssystem-Routinen. Der Stern bei der Adreß-Angabe weist darauf hin, daß es sich nicht um eine sogenannte »Kernal-Routine« handelt, daß also die Adresse ausschließlich für den C 64 gilt.

Name	Adresse	Funktion
AXOUT	$BDCD = 48589*	16-Bitzahl ausgeben
CHKIN	$FFC6 = 65478	Kanal für Eingabe öffnen
CHKKOM	$AEFD = 44797*	Komma holen
CHKOUT	$FFC9 = 65481	Kanal für Ausgabe öffnen
CHRIN	$FFCF = 65487	Zeicheneingabe (siehe unten)
CHROUT	$FFD2 = 65490	Zeichenausgabe
CLALL	$FFE7 = 65511	Alle Dateien und Kanäle schließen
CLOSE	$FFC3 = 65475	Datei schließen
CLRCHN	$FFCC = 65484	Kanäle schließen
CRLF	$AAD7 = 43735*	Return ausgeben
GETIN	$FFE4 = 65508	Zeicheneingabe (siehe unten)
GETPAR	$E1D4 = 57812*	LOAD/SAVE-Parameter holen
LOAD	$FFD5 = 65493	Datei laden
OPEN	$FFC0 = 65472	Datei öffnen
SAVE	$FFD8 = 65496	Speicherbereich speichern
SETNAM	$FFBD = 65469	Filenamen setzen
SETPAR	$FFBA = 65466	File-, Gerätenummer und Sekundäradresse setzen
SPACE	$AB3F = 43839*	Leerzeichen ausgeben

Merke: Bei Eingabe von einem File sind GETIN und CHRIN gleichwertig, in diesem Fall ist aus technischen Gründen GETIN vorzuziehen. Bei Eingabe von Tastatur liest GETIN ein Zeichen aus dem Tastaturpuffer und liefert eine Null, wenn keine Taste gedrückt wurde, während bei CHRIN ein Eingabecursor erscheint und auch längere Texte erfaßt werden können.

Wichtige Speicherzellen (C 64):

$90 = 144	Statusvariable ST
$B7 = 183	Länge des Filenamens
$B8 = 184	logische Filenummer
$B9 = 185	Sekundäradresse
$BA = 186	Gerätenummer
$BB = 187	Adresse des Filenamens LOW
$BC = 188	Adresse des Filenamens HIGH

Multitask - Mehrere Maschinenprogramme gleichzeitig ablaufen lassen

Welcher C64-Programmierer würde es sich nicht wünschen, mehrere Maschinenprogramme gleichzeitig ablaufen lassen zu können? Mit dem hier vorgestellten Tool »Multitask« wird das zum Kinderspiel. Bis zu 31

Assemblerroutinen laufen mit nur leicht verringerter Geschwindigkeit quasi gleichzeitig ab, auf Wunsch kann dabei sogar noch in Basic weiterprogrammiert werden.

von Nikolaus M. Heusler

Die Routine wird mit

LOAD "MT 49152",8,8

geladen. Danach sollten Sie NEW eingeben, um alle Zeiger richtigzustellen. Andernfalls könnte die Fehlermeldung ?OUT OF MEMORY ERROR erscheinen. Als nächstes lädt man die Routinen, die gleichzeitig ablaufen sollen, in den Speicher. Mit dem Befehl

SYS 49152, ad1, ad2, ad3, ad4, ...

werden nun beliebig viele solcher Maschinenprogramme (Minimum 1, Maximum 31) fast gleichzeitig gestartet. Sofern sie sich nicht gegenseitig stören (siehe unten), laufen sie quasi parallel ab.

Die Startadressen der Routinen werden nach dem Komma übergeben: ad1, ad2, ad3 und so weiter. Geben Sie dabei einfach die Adressen an, die sonst auch hinter SYS stehen würden, um die Routinen einzeln zu aktivieren. So würde etwa der Befehl

SYS 49152, 20000, 32000

zwei Maschinenprogramme ab 20000 und 32000 gleichzeitig starten. Wenn in dieser Parameterliste noch das Kaufmannsund (&) steht, kehrt der SYS 49152 nach dem Start der Maschinenprogramme nach Basic zurück. Jetzt kann programmiert werden, während die Assemblerroutinen im Hintergrund laufen (besonders praktisch für Tools und Programmierhilfen oder ähnliches)! Im obigen Fall also etwa:

SYS 49152, 20000, 30000, &

Die Anwendung des & kann beliebig erfolgen, also auch zum Beispiel

SYS 49152, &, 20000, 30000

oder

SYS 49152, 20000, &, 30000

Anstelle der Zahlen können natürlich auch Variablen oder Rechenausdrücke für die Startadressen verwendet werden. Allerdings darf keine der Maschinenroutinen, die gestartet werden sollen, an Adresse 0 beginnen. SYS 49152,...,0,... ist also verboten. Die Routine fängt Fehleingaben ab, es erscheint ein ?ILLEGAL QUANTITY ERROR.

Hier ein praktisches Beispiel für die Anwendung:

```
30000 INC 53280
30003 JMP 30000
```

Startet man dieses Programm mit SYS 30000, flimmert der Bildschirmrahmen in einer Endlosschleife. Soll dieser Effekt auftreten, während Basic läuft,

wenden Sie einfach Multitask an:

SYS 49152,30000,&

Nun könnte gleichzeitig dazu noch ein Sprite horizontal bewegt werden, solange der Feuerknopf eines Joysticks in Port 2 gedrückt wird:

```
35000 LDA #101
35002 STA 53269
35005 STA 53248
35008 STA 53249
35011 LDA #1
35013 STA 2040
35016 LDA 56320
35019 AND #16
35021 BNE 35016
35023 INC 53248
35026 JMP 35016
```

Im Bereich von 35000 bis 35015 wird das Sprite sichtbar auf den Monitor gebracht. Ab 35016 beginnt die Endlosschleife, die für die Bewegung bei <Feuer> zuständig ist.

SYS 49152,35000,30000,&

läßt diese Routine ablaufen - gleichzeitig mit Basic und dem oben gezeigten Flimmern!

Bei den Maschinenprogrammen, die von Multitask ausgeführt werden, handelt es sich also meistens um Endlosschleifen. Dennoch ist eine Funktion eingebaut, die auch Ausstiege mit RTS verkraftet. Näheres dazu weiter unten.

Einige Anregungen für Routinen, die Sie miteinander und zusammen mit Basic aktivieren könnten: Tastenklick, eingeblendete Echtzeituhr, Escape-Funktion (Löschen des Quote-Modus bei besonderer Tastenkombination), verbesserter Editor, Reset auf Tastendruck, ein Sprite, das »von alleine« bewegt wird, während ein Basic-Spiel läuft, und vieles mehr.

Als Beispiel für die Anwendung mag auch das Demo dienen. Es wird mit

LOAD "MT.DEMO",8

geladen und mit RUN gestartet. Das Basicprogramm lädt nun noch das steuernde Maschinenprogramm, die Unterroutinen und einige Spritedaten nach. Sodann erscheint die Frage, ob Basic weiter ablaufen soll. Drücken Sie die Taste <1> oder <2>. Es kann nun einige Sekunden dauern, bis wirklich alle Maschinenroutinen ablaufen, dann erscheint ein galoppierendes animiertes buntes Pferd auf dem Schirm. Es wird von einigen Maschinenroutinen ab 10000 gesteuert:

10000 horizontale Bewegung
10050 vertikale Bewegung
10100 Animation
10150 Pferd sichtbar machen
10200 (interne Warteschleife)
10250 Farbänderung

Bei allen Routinen (bis aus 10150 und 10200) handelt es sich um

Endlosschleifen wie oben erklärt. Jede dieser Routinen kann also mit SYS gestartet werden. Geben Sie SYS 10150 ein, wird nur das Pferd eingeschaltet. Starten Sie jetzt die vertikale Bewegung mit SYS 10050, fährt das Pferd einfach nur nach oben und unten, ohne sonstige Effekte. Das im Demoprogramm erzeugte Sammelsurium wird demnach zum Beispiel mit

SYS 49152, 10150,10000,10050,10100,10250,&

gestartet. Das Zeichen & am Ende bewirkt den Basic-Rücksprung.

Im Prinzip können mit dieser Routine somit beliebige Assemblerroutinen, die an sich Endlosschleifen darstellen, miteinander ablaufen. Sie müssen allerdings aus technischen Gründen einigen Bedingungen entsprechen: Zum einen dürfen keine IRQ-Manipulationen wie SEI oder Verbiegen des IRQ-Vektors vorgenommen werden. Andernfalls kann es passieren, daß etwa der Task, der den Interrupt sperrt, nicht mehr verlassen und als einziger ausgeführt wird. CLI dagegen ist erlaubt. Ebenso ist die Veränderung des Prozessorports (Adresse 1) zu vermeiden. Auch sind natürlich Komplikationen zu erwarten, wenn mehrere Maschinenprogramme die gleichen Speicheradressen verwenden oder gar im selben Bereich arbeiten. Das Utility selbst belegt auch bestimmte Speicherstellen, und zwar den Bereich 49152-49546, $E000-$FFFF unter dem RAM und die Zeropageadressen 2 und 3. Außerdem sollten Operationen mit Peripheriegeräten (Floppy, Drucker, Band) vermieden werden, während das Multitasking aktiv ist.

Mit

SYS 49155

wird der Multitask-Betrieb wieder abgeschaltet. Unter Umständen ist es günstig, die IRQ-Geschwindigkeit und damit die Tast-Wechselgeschwindigkeit zu verändern. Dazu dient der Befehl

SYS 49158,X

X ist ein Wert zwischen 1 und 255, der Wert 0 wirkt wie 256. Je größer X gewählt wird, desto langsamer wechselt der C 64 die Tasks. Lassen Sie den Parameter X weg, wird der Standardwert 60 eingesetzt:

SYS 49158

Bei diesem Utility ist es interessant, zu erfahren, wie ungefähr es intern funktioniert. Genauere Information kann der interessierte Leser dem kommentierten Quelltext entnehmen.

Grundlage für das Multitasking ist der Systeminterrupt (IRQ). Bei jedem IRQ unterbricht der C 64 das momentan laufende Maschinenprogramm, um eine Interruptroutine abzuarbeiten. In diesem Fall handelt es sich um eine neue Routine. Diese holt sich zunächst die Adresse vom Stack, an der das aufrufende Programm unterbrochen wurde, und speichert diese. Einer Tabelle wird die Startadresse des ersten zu startenden Programmes entnommen und auf den Stack geschrieben. Wenn der Computer jetzt die IRQ-Routine verläßt, findet er die neue Adresse auf dem Stack und macht mit dem nächsten Programm weiter - bis zum nächsten IRQ. Auf diese Weise kommen alle gewählten Routinen nacheinander je einmal dran. Danach wird von vorn begonnen. Da die IRQs sehr häufig (normal 60 mal in der Sekunde) auftreten, wird der Eindruck erweckt, die Programme laufen gleichzeitig ab.

Neben der Rettung der Startadressen ist es erforderlich, die drei Prozessorvariablen A, X und Y sowie den Stackpointer, den Stack (Adresse 256 bis 511) und das Statusregister zu speichern. Dazu dient ein reservierter Speicherbereich ab $E000 unter dem Kernal-ROM gemäß untenstehender Speicherbelegung. Dadurch bleibt in den Tasks neben der Möglichkeit, ganz normal alle Prozessor-Register zu benutzen, sogar der Stack voll und ganz erhalten. Befehle wie PLA und PHA dürfen wie gewohnt ausgeführt werden (siehe dazu die untenstehende Bemerkung zum Stack).

Am Ende der IRQ-Routine wird allerdings noch die normale System-IRQ-Routine ausgeführt, damit Effekte wie das Cursorblinken und der Editor weiterhin erhalten bleiben.

Ein Problem stellt sich noch, wenn eines der Maschinenprogramme keine Endlosschleife ist, sondern mit RTS endet. Da es dann keine sinnvolle Rücksprungadresse auf dem Stack finden würde, wäre ein Absturz unvermeidbar. Daher erzeugt das Utility beim ersten Aufruf jeder Routine eine künstliche Rücksprungadresse, die auf eine Endlosschleife zeigt, und schreibt sie auf den Stack. Endet ein Task mit RTS, wird als nächstes diese Endlosschleife ausgeführt. Der Task ist also abgeschlossen, stört aber nicht die Ausführung der übrigen Programme.

Ähnlich funktioniert die Auswahlmöglichkeit, ob das Basic weiter ablaufen soll. Wählen Sie diese Option, kehrt der Befehl SYS 49152 nach der Ausführung einfach mit RTS in den Basic-Editor zurück. Dieser ist jetzt einer der auszuführenden Tasks. Soll kein Basic ablaufen, endet SYS 49152 in einer Endlosschleife.

Vor dem ersten Aufruf jedes Tasks wird außerdem der Stackpointer auf 247 gesetzt, damit das Maschinenprogramm den vollen Stack zur Verfügung hat. Für den Programmierer ist dies jedoch nicht von Bedeutung.

Wir wünschen Ihnen viel Spaß mit diesem neuartigen Multitask-Gefühl. Was Sie alles damit machen können, liegt voll und ganz bei Ihnen und Ihrer Phantasie. Genießen Sie die umfangreichen Möglichkeiten dieser neuen Technik!

Speicherbelegung (hexadezimal):

```
0002-0003  temporär: Zeiger auf Startadressen-Tabelle
c000-c18a  Maschinenprogramm
c000-c008  Sprungtabelle
c009       Init
c01b       Parameterschleife
c086       Endlosschleife, falls Basic ausgeschaltet
c08f       Ende Multitasking
c095       IRQ-Speed setzen
c0a4       Endlosschleife für Task-Ende mit RTS
c0a7       neue IRQ-Routine
c0b0       nächsten Task suchen
c0c2       neue Adresse merken
c0cd       alte Adresse speichern
c0df       neue Register merken
c0fd       alte Register speichern
c119       Stacks vertauschen
c131       ab hier einige Selbstmodifikationen
c139       Zeiger auf nächsten Task
c147       Stackpointer vorbereiten
c14f       Endlosschleife bei RTS vorbereiten
```

c15c   neue Adresse setzen
c166   neue Register setzen
c181   neue Hardware-NMI-Routine
c188   Flag: Basic eingeschaltet
c189   Anzahl Tasks
c18a   Flag: Task bereits einmal aufgerufen
e000-e03f Startadressen-Tabelle
e040-e05f Tabelle: A-Register der Tasks
e060-e07f Tabelle: X-Register der Tasks
e080-e09f Tabelle: Y-Register der Tasks
e0a0-e0bf Tabelle: Status-Register der Tasks
e0c0-e0df Tabelle: Stackpointer der Tasks
e100-feff Stackinhalte der Tasks

Das NSS-Kernal: Runderneuert!

An diesen Beitrag werden Sie vielleicht in naher Zukunft jeden Tag erinnert werden, wenn Sie Ihren Computer einschalten: Aus dem C 64 wird nun doch ein richtiger Profi-Computer! Ein neues Betriebssystem putzt den alten Kasten mächtig heraus. Was halten Sie davon: Directory und andere Floppy- und Druckerfunktionen auf Tastendruck, Renew-Routine, Schnell-Lader mit 16 facher Geschwindigkeit, Funktionstastenbelegung, Renew-Routine und viele Sonderfunktionen für den Programmierer und Anwender.

Das neue Kernal ist eine Hardware-Erweiterung. Dazu muß im C 64 ein IC ausgewechselt werden. Das neue Kernal, das Sie von uns als Datei erhalten, muß wie unten beschrieben in ein Eprom gebrannt werden, welches dann gegen ein ROM im C 64 ausgetauscht wird. Ab jetzt sind die neuen Funktionen sofort nach dem Einschalten des C 64 verfügbar. Sollte es wider Erwarten dennoch einmal Probleme mit anderen Programmen geben, läßt sich diese Low-Cost Erweiterung mit einem Schalter einfach abschalten.

Betriebssystem - Beschreibung

Tastenfunktionen
<CTRL A>   Fastload on/off (wird angezeigt)
<CTRL O>   RENEW + CLR (Anzeige: »RENEW OK«)
<CTRL F>   FloppyIRQ hochsetzen => schnellere Kopfbewegung (Anzeige: »IMPROVED...«), funktioniert unter Umständen nicht gleich beim ersten Mal
<CTRL U>   Die SAVE-Routine speichert nun auch das RAM unter dem Basic-ROM (Anzeige: »SAVE RAM«)
<CTRL K>   Alle neuen Tasten aus; Rückgängigmachen mit <RUN STOP/RESTORE>
<CTRL Z>   Führt ein unterbrochenes LIST fort
<SHIFT/RUN STOP> «LOAD ":*" ,8,1 {RETURN} SYS {RETURN}»
<f1>   Directory ohne Programmverlust
<f3>   Fehlerkanal anzeigen
<f5>   Floppybefehl eingeben (Abschluß mit <RETURN>)
<f7>   «LOAD............,8,1» - Laden aus dem Directory
<f2>   Wie F7, zusätzich <RETURN> und SYS <RETURN>
<f4>   LIST <RETURN>

<f6>   SAVE "
<f8>   RUN <RETURN>
<CBM f5> sendet Text zum Drucker 4, Sek. Adr. 0
<CTRL f5> sendet Text zum Drucker 4, Sek. Adr. 7
Prompts: <f5> alleine: »>«,   mit <CBM>: »]«,   mit <CTRL>: »)«
<CR> am Zeilenende erfolgt automatisch (ohne <LF>)

Wann reagieren diese Tasten nicht? Wenn mindestens eine dieser Bedingungen zutrifft:

· Abgeschaltet (POKE (0),175 oder wegen <CTRL K>)
· Programm-Modus aktiv (PEEK (157)=0)
· INSERT-Modus aktiv (PEEK (216) > 0)
· Hochkommamodus aktiv (PEEK (212) > 0)
· Computer wartet gerade nicht auf Eingabe (Routine ab $E5CA, Veränderung: $E5E7)

Sonstiges

1. Peripheriegeräte:
· Fastload (EXOS-Version) lädt 12-14 mal so schnell von Diskette 1541
· Angabe »,8,1« kann bei LOAD und VERIFY entfallen
· LOAD "NAME",8,0,ADR lädt Programmfile ab ADR
· LOAD "NAME",8 ohne Sekundäradresse lädt absolut (,8,1)
· LOAD zeigt (außer im Programmodus oder bei PEEK (1) ungleich 55) an, wohin geladen wird (dezimal)
· SAVE "Klammeraffe:" funktioniert! (Anzeige: SCRATCHING...) Das Programm wird erst gelöscht (S:) und dann normal gespeichert
· Bei LOAD ,8,1 kann NEW in den meisten Fällen entfallen
· Bei VERIFY wird sofort nach dem ersten Fehler abgebrochen
2. Bedienung des C 64:
· Bildschirmfarben: Rahmen und Hintergrund dunkelgrau, Schrift weiß
· Cursorgeschwindigkeit erhöht
· Tastenrepeat beim Einschalten für alle Tasten (PEEK (650)=235)
3. Basic-Programmierung:
· Kein Listschutz mehr wirksam (Ausnahme: Nullbyte nach Zeilennummer)
· <SHIFT> hält Scrollen an, <CTRL> verlangsamt nicht mehr so stark
· SYS ohne Zahl startet das letztgeladene Programm an der Adresse, an die es geladen wurde (oder mit RUN, wenn Adresse = DEEK (43)). Die Ladeadresse merkt sich das System in DEEK (671) (ehemals IRQ-Vektor bei Cassettenbetrieb)
· Durch Druck auf <CTRL> und <RESET> können Sie ein Maschinenprogramm über den Vektor 671/2 starten (z.B. Spritekiller möglich!)
4. Hardware:
· <RESET>: Gleichzeitig mit <SPACE>: <RESET>-Schutz überlistet
                            <C=>:   Klingeln ertönt (SID-Test)
                            <CTRL>: Autostart (siehe oben)
           RAM-Test entfällt (Reset wesentlich schneller)
           CIAs programmiert auf 50 Hz (Routine ab $FDA3)
· <RESTORE> + <SPACE> überlistet <RESTORE>-Schutz
· Grafik ab $E000 wird bei <RESET> oder <RESTORE> nicht zerstört
· Versionsnummer (PEEK(65408)) = 66
5. Korrektur von Systemfehlern:
· Bildschirmlöschroutine problemlos
· Das Bildschirmlöschen geht etwas schneller
6. Einschränkungen:
· Tape kann nicht mehr benutzt werden
· WICHTIG: Kein VERIFY bei aktiviertem Fastload möglich

- Die RS 232-Schnittstelle kann nicht mehr benutzt werden
- Das Directory kann nicht mit dem Schnell-Lader geladen werden. Es sollte nur mit <F1> geladen werden.

Interna

```
POKE 0, PEEK (0) OR 128    Tasten aus
POKE 0, PEEK (0) OR 64     Fastload aus
POKE 0, PEEK (0) AND 127   Tasten an
POKE 0, PEEK (0) AND 191   Fastload an
<RUNSTOP RESTORE> schaltet beide Funktionen an
Also: PEEK (0): Bit 7: Tasten   (1 = aus, 0 = an)
                Bit 6: Fastload (1 = aus, 0 = an)
```

Einbau

Das entsprechende File ist auf ein Eprom vom Typ 27128 zu brennen (Eprom-Brenner). Dieses Eprom ist in einen handelsüblichen Adaptersockel zu stecken (zum Beispiel von REX Datentechnik, Hagen, Bestellnummer 9598), der dann anstelle des Kernal-ROMs (Bezeichnung U4) in den C 64 (alte Version) eingebaut wird. U4 findet sich, wenn Sie den C 64 aufklappen, als der mittlere der drei kleineren 24-poligen Chips in der oberen Reihe links unter dem Cassettenport und rechts vom Userport. Falls das System in ein 27128 gebrannt wird, soll das neue System in der unteren Bank 0 ($0000-$1FFF) und das Original-Kernal in Bank 1 (oben, $2000-$3FFF) stehen. Der Einbau sollte nur vom Fachmann vorgenommen werden! Das NSS-Kernal eignet sich nicht zum Betrieb in einer ROM-Steckkarte für den Expansion-Port, da hier die Einblendung in den Bereich ab $8000 erfolgt. Für den Einbau in einen neuen C 64 muß zusätzlich noch der Basic-Interpreter in das Eprom 27128 gebrannt werden, ein Umschalten ist dann nicht ohne weiteres möglich.

Brennen des Eproms

Um das Kernal in ein handelsübliches Eprom zu brennen, benötigen Sie neben dem Eprom 2764 einen geeigneten Eprom-Brenner. Jetzt wird das File »NSS KERNAL 2764« unter Beachtung der Hinweise in der Anleitung zum Brenner in das Eprom gebrannt. Soll bei Bedarf auf das alte Betriebssystem umgeschaltet werden können, verwenden Sie ein Eprom vom Typ 27128 und brennen in dieses das File »NSS KERNAL 27128«. Zur Umschaltung beachten Sie die Hinweise in der Anleitung zum Adaptersockel.

Sicherheitsvorschriften für CMOS-ICs

CMOS-ICs sind sehr empfindliche elektronische Bausteine. Daher sollten sie mit besonderer Vorsicht behandelt werden. Diese Chips sind unter anderem sehr empfindlich gegen statische Aufladung, daher sollten Sie Berührungen vor allem der elektrischen Anschlüsse (»Beinchen«) vermeiden. Der schwarze Stoff (Leitgummi) schützt gegen Aufladungen, er muß jedoch vor dem Betrieb entfernt werden. Auch sollten diese Bausteine keinen extremen Temperaturen oder Schwankungen ausgesetzt werden. Ebenfalls sind größere mechanische Belastungen (z.B. Stöße, Reibung) zu vermeiden. Das IC darf nur bei ausgeschaltetem Rechner eingesetzt werden.

Eproms werden durch UV-Licht gelöscht. Daher sollten Sie nicht den Schutzaufkleber abziehen, und intensive Sonnenstrahlung oder die Bestrahlung durch UV-Lampen, etwa Höhensonnen, vermeiden.

Bestell-Service

Sollte es Ihnen zu mühsam oder nicht möglich sein, die Datei in ein Eprom zu brennen, können Sie beim Verlag ein einbaufertiges gebranntes Eprom komplett mit Adapterplatine erhalten. Das Bauteil muß dann nur noch an Stelle des alten ROMs in den C 64 eingesetzt werden. Der Preis für den Zusatz beträgt DM 79,- (zahlbar im Voraus in bar oder per Scheck oder bequem per Nachnahme). Geben Sie bei der Bestellung bitte an, ob Sie einen neuen oder alten C 64 besitzen.

Die Adresse:

IPV Ippen & Pretzsch Verlags GmbH
Pressehaus Bayerstraße 57-59
8000 München 2
Tel. 089/8542412
Fax. 089/8545837

Dieses Angebot ist leider nicht mehr gueltig. Der IPV Verlag ist nicht mehr unter dieser Anschrift erreichbar.

Hinweis: Der Einbau des Bauteils in Ihren C 64 ist nur möglich, falls das Kernal-ROM (Position U4 beim alten und neuen C 64) gesockelt ist, da man sonst das alte IC nicht ohne weiteres entfernen kann! Falls nicht, müssen Sie das alte Original-IC auslöten und durch einen Sockel ersetzen. Bitte beachten Sie, daß durch diesen Eingriff jegliche Garantieansprüche am Computer verlorengehen.

Jetzt wird's kalt: Eisberg

Ganze 32 Fehler- und Störungsmeldungen hält der C 64 für den Programmierer bereit. Aus eigener Erfahrung wissen Sie, wie ärgerlich es ist, wenn einen der Computer so besserwisserisch und kalt abserviert. Aber Sie sitzen am »längeren Hebel«! Schalten Sie die Fehlermeldungen doch einfach ab - mit »Eisberg«. Man kann eingeben, was man will, es erscheinen einfach keine Fehlermeldungen mehr. Dieses nur drei Blocks kurze Gag-Programm hat jedoch durchaus auch praktischen Nutzen. Machen auch Sie Ihre Programme professioneller!

Laden Sie das Programm »Eisberg« (oft ist es gar nicht so einfach, einen guten Namen für ein neues Programm zu finden) mit dem Befehl

```
LOAD "EISBERG",8
```

Der Start erfolgt mit RUN, Assembler-Kenntnisse sind also zur Anwendung dieses Maschinenprogramms nicht notwendig. Eine Einschaltmeldung erscheint. Ab jetzt sind alle Basic-Fehlermeldungen sowie die Störungs-Hinweise ?REDO FROM START und ?EXTRA IGNORED bei INPUT-Befehlen abgeschaltet. Sie können eingeben, was Sie wollen, es erscheint einfach keine Fehlermeldung mehr:

```
PRINT 1/0
HALLO
A = SIN
```

```
TI$ = "ANDREAS"
RETURN:RETURN
NEXT
1 GOSUB 1
PRINT FNX(0)
POKE -1,4834
PRINT 1E99
LOAD "$",30
PRINT VAL(18)
A$ = 8
CONT
DIM O(10),O(11)
PRINT O(1000)
INPUT
PRINT ASC("")
GOTO 65000
```

Bei allen diesen Befehlen wirft der Computer sonst gnadenlos irgend eine Fehlermeldung aus. Mit aktiviertem Eisberg wird der »defekte« Befehl entweder einfach gar nicht ausgeführt, sondern übersprungen, oder der C 64 gibt ein unsinniges Ergebnis aus.

In einigen Fällen - und darin liegt der praktische Wert von diesem Hilfsprogramm - wird der Befehl sogar »kulant« ausgeführt, der Computer nimmt einen anderen Befehl an, dessen Wirkung der vom Programmierer gewollten nahekommt. Beispiel GOTO-Befehl: Beziehen Sie sich mit GOTO 90 auf eine nicht existierende Programmzeile, sucht sich das Programm die Zeile mit der nächst höheren Nummer und springt diese an. Ist im Beispiel die Zeile 90 nicht existent, aber dafür Zeile 100, so geht es in Zeile 100 weiter (oder in der nächsten Zeile, die im Programm auf 90 folgt).

Ein weiterer ärgerlicher Fehler, der in Basic manchmal auftritt, ist der ?ILLEGAL QUANTITY ERROR, den die ASC-Funktion auf einen Leerstring angewandt ausgibt. Beim Lesen aus Dateien werden aber für Nullbytes leere Strings erzeugt. Die ASC-Funktion wurde daher so umgebaut, daß ein leerer String den ASCII-Code Null hat. Komplizierte Konstruktionen wie

```
10 GET#2,A$
20 A=ASC(A$+CHR$(0))
```

um eine Datei zu analysieren können Sie sich in Zukunft sparen.

Oder es soll mit GET A eine Menüabfrage realisiert werden. Gewöhnlich wird die Betätigung einer alphanumerischen Taste durch einen ?SYNTAX ERROR quittiert, nicht aber mit Eisberg.

Interessante mathematische Effekte lassen sich beobachten. Die Teilung durch Null, mathematisch als Unendlich definiert, ergibt beim C 64 eine sehr große Zahl, also sogar fast das korrekte (?) Ergebnis. Die Logarithmus-Funktion ist nur für positive Zahlen definiert. Gibt man einmal PRINT LOG(0) ein, erscheint kein ?ILLEGAL QUANTITY ERROR mehr, sondern eine negative Zahl. Wird diese als Argument der Umkehrfunktion EXP gesetzt, ist das Ergebnis fast Null. Im Grunde genommen liegt also nur ein Rundungsfehler vor. Ähnliche Effekte treten bei Anwendung der transzendenten Funktionen auf.

In anderen Fällen arbeitet das Programm zwar fehlerhaft, aber es steigt nicht aus. Beispielsweise erscheint die ?STRING TOO LONG-Meldung bei zu langen Texten nicht mehr, es werden einfach alle Zeichen ab Nr. 256

abgeschnitten. Bei zu hohen Werten etwa hinter POKE verwendet der Computer nur die Bits, die ihn interessieren. POKE 4,900 wirkt also wie POKE 4,132, da das Lowbyte von 900 gerade 132 ist. Gleiches gilt für die Adreßangabe bei SYS, POKE, WAIT und PEEK.

Die Liste läßt sich beliebig fortsetzen. RETURN ohne GOSUB, zu komplizierte Verschachtelungen, falsche oder fehlende Parameter, Dateifehler, fehlende Doppelpunkte oder Kommas, nichts bringt den Computer mehr aus der Ruhe. Fertige und ausgetestete (!) Programm werden so noch professioneller, da im Falle eines Bedienungs- oder übersehenen Programmierfehlers kein Ausstieg mit einer Fehlermeldung mehr erfolgt.

Es soll betont werden, daß alle Funktionen des C 64 absolut unverändert normal arbeiten, wenn keine Fehler auftreten. Am Lauf eines fehlerfreien Programms oder Algorithmus wird sich also rein gar nichts ändern, wenn Eisberg aktiviert ist.

Eisberg bleibt auch nach einem Warmstart mit <RUN STOP/RESTORE> aktiv. Es läßt sich nur mit einem Reset oder dem Befehl

POKE 1,55

ausschalten. Danach erfolgt der Neustart, falls sich das Steuerprogramm noch im Speicher befindet, mit

SYS 51200

Zur Erklärung der internen Vorgänge: Nach dem Laden und Starten kopiert sich Eisberg in einen Bereich von 51200 bis 51711 ($C800 bis $C9FF) und startet sich dort. Jetzt werden das Basic- und das Betriebssystem-ROM in die darunterliegenden RAMs kopiert. Dort nimmt das Programm umfangreiche Änderungen vor. An vielen Stellen werden Branch-Befehle, die bei Störungen eine Fehlermeldung auslösen, unwirksam gemacht. Manche Routinen, die auf Fehlbedienung prügen, werden einfach mit einem RTS unwirksam gemacht. An einigen Stellen baut Eisberg Sprungbefehle ein, die in eine Routine münden, die das Ende des aktuellen Basicbefehle suchen (DATA-Befehl) und dann die Interpreterschleife aufrufen. Diese Routine ab $A19E (dort standen vorher die Original-Fehlermeldungstexte) bewirkt, daß ein fehlerhafter Befehl einfach nur übersprungen wird, es erfolgt kein Programmabbruch. Diese Technik hat gegenüber der Methode, einfach die Zentralroutine zur Ausgabe einer Fehlermeldung auf den Basicstart zu verbiegen, den Vorteil, daß ein fehlerhaftes Programm weder mit noch ohne Meldung beendet, sondern einfach beim nächsten Befehl fortgesetzt wird.

Nach den Änderungen schaltet Eisberg mit POKE 1,53 die ROMs aus und die RAMs ein. Dort befinden sich jetzt modifizierte Versionen von Kernal und Basic, die ab sofort ablaufen.

Der Bereich ab 51200 wird jetzt nicht mehr benötigt und steht ebenso wie der Basicspeicher anderen Anwendungen offen.

Absturz? Nein Danke! »Uncrash« hilft

Computer abgestürzt, Programm futsch? Selbst schuld! Hätten Sie vorher »Uncrash« aktiviert, hätten Sie das Schlimmste vielleicht verhindern können. Sowohl Basic- wie auch Maschinenprogrammierer werden dieses kleine Utility besonders schätzen, das den C 64 praktisch absturz-steril macht. Mit diesem Programm werden Sie, wenn überhaupt, nur noch selten wegen eines Absturzes Daten verlieren. Enthalten ist eine Renew-Routine, die auch die Variablenwerte wiederherstellt.

Einer der großen Vorzüge des C 64 gegenüber anderen Homecomputern ist die Möglichkeit, ihn völlig an die eigenen Bedürfnisse anzupassen. Mit seinen Vektoren und dem Prozessorport haben Sie als Programmierer weitgehenden Einfluß auf die internen Vorgänge. Diese offene Architektur macht den Computer allerdings auch sehr verwundbar gegen falsche Programmierung. Ein einziges falsches Kommando und Totenstille kehrt ein. Manchmal arbeitet das Gerät nach <RUN STOP/RESTORE> weiter - aber nicht immer. Sie können den Rechner aus- und wieder einschalten, das hilft garantiert, löscht aber ebenso sicher die gespeicherten Daten, inklusive Ihr Programm samt seiner Variablen und Tools, mit denen Sie vielleicht arbeiten. Das kann schon sehr frustrierend sein.

Manche C 64-Besitzer haben mittlerweile einen Reset-Taster installiert, mit dem ein echter Kaltstart durchgeführt werden kann. C 128-Anwender haben einen solchen Knopf »ab Werk« am Computer. Ob Sie diesen Zusatz Ihr eigen nennen oder nicht, Sie profitieren in jedem Fall von »Uncrash«.

Es handelt sich dabei um ein kurzes Maschinenprogramm. Sie brauchen jedoch keinerlei Assemblerkenntnisse, um damit arbeiten zu können. Das Programm ist in der Lage, bestimmte Abstutzsituationen selbstständig zu erkennen und zu verhindern. Es erscheint unglaublich, aber Uncrash erkennt sogar bestimmte Basicbefehle, die einen Absturz bewirken würden und führt sie gar nicht aus. Sollte sich der Computer wirklich einmal aufgehängt haben, gibt es auf Tastendruck zwei Stufen, wieder herauszukommen.

Einzige Bedingung ist, daß Sie Uncrash vor Beginn der Session installiert haben. Dazu laden Sie das Programm mit

```
LOAD "UNCRASH",8
```

Obwohl das Programm aus Gründen des Koforts und der Geschwindigkeit vollständig in Maschinensprache verfaßt ist, brauchen Sie keine Assemblerkenntnisse, um damit arbeiten zu können. Die Bedienung erfolgt wie gewohnt in Basic. Starten Sie nach dem Laden das Utility mit RUN. Es ist jetzt aktiviert, eine Meldung erscheint. Soll das Tool geladen und aktiviert werden, während bereits ein Basicprogramm im Speicher steht, das bei obigem Manöver natürlich verloren gehen würde, besteht auch die Möglichkeit, Uncrash mit

```
LOAD "UNCRASH 52809",8,1
```

absolut von Diskette zu laden. Nachdem Sie dann mit NEW alle Zeiger wieder richtiggestellt haben, aktivieren Sie das Tool mit

```
SYS 52809
```

Die Meldung »UNCRASH« erscheint auch diesmal, und der Bildschirm ändert

seine Farben. Als kleiner Bonus werden diese Farben auch später nach jedem <RUN STOP/RESTORE> erscheinen, Sie können also mit

```
POKE 53106, Cursorfarbe
POKE 53111, Rahmenfarbe
POKE 53116, Hintergrundfarbe
```

Ihre persönlichen Lieblingsfarben einstellen.

Nach der Aktivierung schlummert das Hilfsprogramm vor sich hin, bis Sie ihm mitteilen, daß der Computer abgestürzt ist. Tun Sie das, indem Sie die RESTORE-Taste drücken. Oft reicht das aus, um den Computer wieder zum Leben zu erwecken. Manche Probleme sind indes so gravierend, daß »härtere« Maßnahmen vonnöten sind: Der Kaltstart. Bei Uncrash gibt es eine spezielle Tastenkombination, die den Kaltstart auslöst. Halten Sie die Tasten <1>, <2> und <Pfeil nach links> gleichzeitig gedrückt und betätigen dann <RESTORE>. Diese vier Tasten zusammen gedrückt bewirken erstens, daß Uncrash sich selbst abschaltet. Zum zweiten wird das Basicprogramm im Speicher gelöscht. Der Computer löst einen Reset aus. Um nun sowohl Uncrash wie auch Ihr Programm wiederherzustellen, geben Sie einfach nochmals

```
SYS 52809
```

ein. Sie werden feststellen, daß das Basicprogramm mitsamt all seinen Variablen und Feldern wieder da ist. Jetzt könnte man zum Beispiel die Variablenwerte auf dem Bildschirm ausgeben und versuchen, damit festzustellen, wodurch der Absturz zustandekam. Möchten Sie das Programm nicht wiederherstellen, definieren Sie nach dem Kaltstart vor dem SYS-Befehl einfach eine beliebige Basic-Variable, etwa

```
X = 1
```

Dies kann dann erforderlich sein, wenn vermutlich verstellte Basiczeiger den Hänger bewirkt haben.

Die Renew-Funktion ist auch allein verfügbar. Dazu geben Sie einfach

```
SYS 52812
```

ein.

Wir wollen nun einmal das Tool einem harten Test unterziehen. Die folgenden POKE-Befehle bewirken bei einem ungesicherten C 64 garantiert einen Absturz, aus dem man so einfach nicht mehr herauskommt. Ist jedoch Uncrash aktiviert, werden einige der fatalen POKEs gar nicht ausgeführt. Ist der Computer dennoch abgestürzt, kommen Sie wie beschrieben ganz einfach aus der Mißlage heraus:

```
POKE 1,51      I/O abschalten
POKE 1,48      BASIC abschalten (wird nicht ausgeführt)
POKE 115,0     Alle Befehle abschalten
POKE 648,0     Tastatur abschalten
POKE 770,128   Direktmodus abschalten
POKE 772,121   BASIC aufhängen
POKE 56322,0   Tastatur abschalten
```

Wichtig ist, daß Sie niemals die RESTORE-Taste während eines Kaltstarts drücken. Das könnte sonst die Ursache für einen Absturz sein, den nicht

einmal mehr Uncrash beheben kann. Es gibt noch eine Einschränkung. Uncrash ist leider nicht in der Lage, Abstürze zu beheben, die auftreten, wenn der Prozessor auf illegale OP-Codes trifft. Es gibt keine Möglichkeit, einen solchen Absturz zu verhindern oder rückgängig zu machen. Wenn Sie natürlich einen Reset-Taster Ihr eigen nennen, können Sie diesen betätigen und danach mit

```
SYS 52809
```

das verlorene Programm zurückholen.

Noch ein technischer Hinweis: Das Programm belegt keinerlei Basic-Speicher, sondern arbeitet im Bereich von $CE49 bis $CFFF (52809 bis 53247). Hier befindet sich das Programm mit seinen Variablen ab $CFE2. Es werden keinerlei andere Speicherzellen belegt.

Insgesamt bietet Uncrash einen gehörigen Schutz vor Störungen, ohne den Sie nicht mehr das Haus verlassen sollten!

Anleitung zum »Quickie-Generator«

Dieses (völlig jugendfreie) Programm dient dazu, Daten im Speicher (z.B. Grafikdaten oder Maschinenprogramme) in einen Basic-Data-Lader zu konvertieren. Die fertigen Programme können ganz normal unter Basic mit LOAD geladen und mit RUN gestartet werden, das Merken umständlicher SYS-Befehle entfällt. Dabei werden die Daten nicht numerisch gespeichert, sondern es wurde ein guter Kompromiß getroffen, damit das erzeugte Programm kurz ist, schnell arbeitet und leicht abzutippen ist. Der Generator eignet sich daher dazu, fertige kurze Maschinenprogramme in 20-Zeiler für den gleichnamigen Wettbewerb des 64'er-Magazins zu konvertieren. Aber auch viele andere Anwendungen lassen sich finden.

Der »Quickie-Generator« wurde mit sich selbst behandelt. Daher kann er einfach mit

```
LOAD "QUICKIE-GENERATO",8
RUN
```

aktiviert werden. Damit haben Sie die 15-Zeiler-Version dieses Programmes geladen und gestartet. Für die Anwendung in der Praxis eignet sich allerdings besser die Version, die absolut auf Diskette gespeichert ist, sie wird mit

```
LOAD "QUICK-GEN $7000",8,8
NEW
```

geladen. In jedem Fall sollten Sie jetzt die zu konvertierenden Daten in den Speicher laden. Zur Sicherheit geben Sie vorher ein:

```
POKE 56,112:CLR
```

damit der Generator nicht von Daten überschrieben wird. Er belegt den Speicher von 28672 bis 29101 einschließlich. In der Zeropage werden die Zellen 2 bis 4, 181, 182 sowie 247 bis 253 belegt.

Der Generator kann jetzt, nachdem Sie sicherheitshalber mit NEW den Basicspeicher leergefegt haben, mit

```
SYS 28672,A,E,S,Z,V
```

aktiviert werden. Dabei gibt A die Anfangsadresse des zu konvertierenden Bereiches an, E die Endadresse (einschließl.), S die Startadresse, mit der das konvertierte Maschinenprogramm gestartet werden kann, Z die erste Zeilennummer und V die Schrittweite der erzeugten Basic-Zeilennummern. Sollen die konvertierten Daten nicht mit SYS gestartet werden, setzen Sie für S irgend einen Wert ein und löschen dann in der zweiten erzeugten Zeile des SYS-Befehl, bevor Sie den »Quickie« speichern. A und müssen folgender Bedingung entsprechen:

```
0 <= A <= E <= 65535
```

Während der Konvertierung sehen Sie, wie das Programm am Bildschirm Zeile für Zeile aufgebaut wird. Bitte drücken Sie während dieser Phase auf keinen Fall eine Taste, sondern warten, bis die Konvertierung abgeschlossen ist. Danach erscheint ein Hinweis, wie viele Zeilen erzeugt wurden (wichtig für den 20-Zeiler Wettbewerb: Es dürfen nicht mehr als 20 Zeilen sein) und der Vermerk »OK«. Jetzt steht das erzeugte Programm im Speicher, es kann bei Bedarf weiter bearbeitet oder auf Diskette gespeichert werden.

Zum Schluß noch einige kurze Anmerkungen zur internen Funktionsweise. Die Daten werden in High- und Lowbyte zerlegt und in Buchstaben von A (Nibblewert 0) bis P (15 = $F) gewandelt. Die Buchstaben für High- und Lowbyte werden direkt hintereinander in die Data-Zeile geschrieben, von wo sie dann später von der READ-Schleife, die der Generator ebenfalls erzeugt, gelesen werden. Die einzelnen Zeilen werden erst auf dem Bildschirm aufgebaut und dann über die Tastaturpuffer-Methode in den Basicspeicher übernommen. Dadurch können mit dem »Quickie-Generator« auch bereits bestehende Basicprogramme um die Data-Zeilen ergänzt werden, sie müssen nur vor der Konvertierung ganz normal geladen werden. Damit der Tastaturpuffer abgefragt wird, muß der Generator kurzzeitig in den Direktmodus zurückkehren. Danach wird mit Hilfe der USR-Funktion wieder zurück in Assembler gesprungen. Der einfachere Weg (SYS-Befehl) wurde nicht gewählt, damit der Generator in seinem Quelltext einfach an neue Startadressen verschoben werden kann (bei SYS wäre das numerische Argument im Klartext notwendig gewesen, was Probleme bei der Assemblierung bereitet hätte, während bei USR die Adresse als Low/Highbyte übergeben wird). Sie können übrigens nach Abschluß der Konvertierung mit dem Befehl

```
J=USR(0)
```

die Anzahl der erzeugten Zeilen noch einmal abrufen.

Der »Quickie-Generator« eignet sich dazu, Maschinenprogramme mit einer Länge bis 648 Bytes in Programme zu konvertieren, die den Bedingungen des 64'er Wettbewerbs »20-Zeiler« entsprechen: Maximal 20 Basic-Zeilen, die von einem ausgdruckten Listing ohne Tricks abgetippt werden können (falls Zeilennummern über 999 Verwendung finden, muß der Data-Befehl mit D <SHIFT A> abgekürzt werden).

Floppy - Tip: Neues DOS

Beim Einschalten führt die 1541 jedesmal eine unnötige Kunstpause aus, während der Sie dem Computer nicht zur Verfügung steht. Das muß nicht sein: Nur einige kleine Änderungen im Betriebssystem der Station sorgen für Abhilfe.

Das Laufwerk 1541 muß bei jedem Reset (also auch beim Einschalten) erst etwa zwei Sekunden arbeiten, bis es ansprechbar ist (man erkennt dies daran, daß es so lange dauert, bis der Motor und die rote LED abgeschaltet werden). Der Grund für diese »Kunstpause« ist, daß hier wie auch beim C 64 ein umfangreicher Test durchgeführt wird, bei dem jede einzelne RAM-Speicherzelle sowie die beiden ROMs auf Funktionsfähigkeit überprüft werden. Dazu wird eine Prüfsumme über alle ROM-Adressen gebildet und mit einem Sollwert verglichen. Das Ärgerliche dabei ist, daß der C 64 abstürzt, wenn Sie versuchen, während dieser Testphase unmittelbar nach dem Einschalten auf das Laufwerk zuzugreifen.

Alle diejenigen, die ROM- oder RAM-Defekte des Laufwerkes nur aus der Literatur kennen -- das werden die meisten sein -- können mit geringen Hardware-Kenntnissen diesen Umstand verbessern, indem sie das DOS so ändern, daß die Reset-Testroutine entfällt. Als Bonbon findet sich im so freigewordenen Bereich jetzt eine Routine, die auch einen Selbsttest durchführt, allerdings -- viel wichtiger, weil störungsanfälliger -- für die rote LED und die beiden Motoren. Dieser Test kann, wenn gewünscht, auch ohne den C 64 beim Einschalten der Station durch Öffnen eines Kontaktes auf der Floppyplatine oder aber nach dem Einschalten des Laufwerkes durch Senden eines Befehles vom C 64 aus geschehen.

Um in den Genuß der Erweiterung zu kommen, müssen Sie zunächst das originale Floppy-DOS von $e000 bis $ffff auslesen. Sehr bequem geht dies mit dem Programm »ROM-Klau« (64'er Sonderheft 57). Die so erzeugte Kopie muß im C 64 von 8192 ($2000) bis 16383 ($3fff) liegen. Laden Sie nun einen Maschinensprachemonitor, der diesen Bereich nicht belegt (z.B. den SMON). Mit diesem werden jetzt im RAM die notwendigen Änderungen vorgenommen. Das folgende Assemblerlisting gibt die Befehle an, die Sie jetzt mit dem Assembler eingeben. Beachten Sie die Adressen. Dieses Listing können Sie sich übrigens auch dann durchlesen, wenn Sie die Änderungen nicht durchführen wollen. Wenn Sie sich für die Floppyprogrammierung interessieren, könnte es einen gewissen »Lehreffekt« haben. Gerade die Programmierung des Steppermotors ist ein leidiges Kapitel.

Folgende Änderungen sind durchzuführen:

```
          ; NEUES FLOPPYSYSTEM
          ; $2/3xxx (im C 64) wird zu $e/fxxx (in der Floppy)

.$2aa7 inx         ; gibt Null
.$2aa8 txa         ; für A auch
.$2aa9 sta   00,x
.$2aab sta 0100,x  ; Zeropage löschen
.$2aae sta 0200,x
.$2ab1 inx
.$2ab2 bne 2aa9
```

```
.$2ab4 lda 1800    ; Geräteadresse (Taster) prüfen
.$2ab7 and  #40    ; Bit gesetzt ? Wenn ja, Taster gedrückt
.$2ab9 beq 2b22    ; nein, weiter mit alter Routine
         ; Selbsttest-Routine
.$2abb sei         ; IRQ sperren
.$2abc lda  #6f    ; DDR: ACT, SYNC = Eingang
.$2abe sta 1c02    ; in DDR setzen
.$2ac1 jsr ead4    ; siehe unten $2ad4
.$2ac4 lda  #0c    ; Hauptmotor und LED anwerfen
.$2ac6 sta 1c00    ; (wichtig für Steppermotor)
.$2ac9 jsr ff2f    ; Steppermotor testen
.$2acc lda 1800    ; Taster wieder testen
.$2acf eor  #40    ; Bit umdrehen:
.$2ad1 jmp eab7    ; wenn nicht wieder gedrückt, weitermachen

        ; Subroutine: Schreibschutz testen
.$2ad4 ldx  #ff    ; Pausenwerte
.$2ad6 ldy  #ff
.$2ad8 lda  #05    ; gibt ca. fünf Sekunden
.$2ada sta 0500    ; unbenutzte Speicherzelle, dient als Zähler
.$2add lda 1c00    ; Port lesen
.$2ae0 and  #10    ; ACT-Bit isolieren, alle anderen Null
.$2ae2 lsr         ; und zum LED-Bit machen
.$2ae3 sta 1c00    ; zurück in Port
.$2ae6 dex         ; weiter
.$2ae7 bne 2add
.$2ae9 dey
.$2aea bne 2add
.$2aec dec 0500    ; nächste Sekunde
.$2aef bne 2add    ; schon fünf Sekunden ? nein, weiter

         ; LED und Hauptmotor blinken
.$2af1 ldx  #10    ; 16 ($10) mal blinken
.$2af3 lda  #00    ; Motor und LED aus
.$2af5 sta 1c00    ; beide blinken zunächst gleichphasig
.$2af8 lda 1c00    ; Port lesen
.$2afb eor  #0c    ; LED und Motorbit umdrehen
.$2afd sta 1c00
.$2b00 ldy  #00    ; kurze Pause (ca. 0,5 Sek.)
.$2b02 lda  #00
.$2b04 asl  500    ; Zeit schinden
.$2b07 iny
.$2b08 bne 2b04    ; Warteschleife
.$2b0a clc
.$2b0b adc  #01
.$2b0d bne 2b04
.$2b0f dex         ; nochmal blinken ?
.$2b10 beq 2b20    ; nein, RTS
.$2b12 cpx  #08    ; schon achtmal geblinkt ?
.$2b14 bne 2af8    ; nein, weiter
.$2b16 lda 1c00
.$2b19 eor  #04    ; Motor-Bit umdrehen
.$2b1b sta 1c00    ; Motor und LED blinken jetzt wechselseitig
.$2b1e bne 2af8    ; unbedingter Sprung
.$2b20 rts         ; genug geblinkt
.$2b21 nop         ; unbenutzt
.$2b22 ldx  #45    ; ab hier weiter wie im Original
```

```
          ; Subroutine: Steppermotor testen
.$3f2f lda  #04    ; viermal testen
.$3f31 sta 0500    ; dient als Zähler
.$3f34 lda  #b0    ; 40 Tracks nach außen
.$3f36 sta   4a    ; Befehl an Kopfsteuerung
.$3f38 jsr ff50    ; Steppermotor bewegen, kurze Pause
.$3f3b lda   4a    ; Stepper schon plaziert ?
.$3f3d bne 3f38    ; nein, weiterbewegen
.$3f3f lda  #50    ; sonst 40 Tracks nach innen
.$3f41 sta   4a    ; Befehl an Kopf
.$3f43 jsr ff50    ; Steppermotor bewegen, warten
.$3f46 lda   4a    ; schon fertig ?
.$3f48 bne 3f43    ; nein, gleich nochmal
.$3f4a dec 0500    ; nochmal testen ?
.$3f4d bpl 3f34    ; ja, noch nicht viermal
.$3f4f rts         ; zurück in Reset-Routine

          ; Subroutine: Steppermotor bedienen, kurze Pause
.$3f50 jsr fa2e    ; Steppermotor bewegen
.$3f53 ldy  #f8    ; Steppergeschwindigkeit
.$3f55 ldx  #ff
.$3f57 dex
.$3f58 bne 3f57    ; Warteschleife
.$3f5a iny
.$3f5b bne 3f57
.$3f5d rts

.$3279 lda  #10    ; Timer 1 High, war ursprünglich $3a
                   ; ergibt Erhöhung der IRQ-Geschwindigkeit
```

Im Listing sind alle Adressen, Parameter und Daten hexadezimal angegeben. Die letzte Änderung an $3279 bewirkt, daß im Laufwerk die IRQ-Frequenz stark erhöht wird, was zum Beispiel beschleunigte Lese- und Schreibvorgänge zur Folge hat. Vor allem aber wird jetzt der Steppermotor sehr viel schneller bewegt, was Zeitgewinn unter anderem bei Befehlen wie Scratch oder Validate zur Folge hat.

Wollen Sie nur den Reset schneller machen, sich aber den Selbsttest sparen (weniger abzutippen), sollten Sie an Adresse $2ab4 einen jmp $eb22 einbauen und den Teil von $2ab7 bis $2b21 einschließlich nicht eingeben. Sie können ihn mit NOPs füllen, oder die alte ROM/RAM-Testroutine dort stehen lassen.

Damit die ganze bisherige Arbeit nicht umsonst war, wäre es nicht falsch, die Änderungen jetzt auf ein Eprom zu brennen. Nachdem alle Manipulationen beendet sind, speichern Sie, auch mit dem Monitor, den Bereich von 8192 ($2000) bis 16383 ($3fff) auf Diskette. Dieses 33 Blocks lange File wird mit einem Eprommer (z.B. Tiny-Eprommer, 64'er 8/88) auf ein Eprom vom Typ 2764 gebrannt. Dieses stecken Sie in einen der üblichen Adaptersockel (z.B. Rex-Datentechnik 9598 oder Conrad Electronic, Best. Nr. 983179), der anstelle des dort befindlichen ROM Bausteins in den Sockel UB4 rechts hinten in der Diskettenstation eingebaut wird.

Wenn Sie jetzt das Laufwerk einschalten, läuft der Motor nur ganz kurz an und geht zusammen mit der roten LED sofort wieder aus, die 1541 ist betriebsbereit.

Um jetzt »manuell« den Selbsttest auszulösen, starten Sie die Routine ab $eabb:

```
OPEN 1,8,15,"M-E"+CHR$(187)+CHR$(234)
```

Danach kann das Laufwerk vorerst vom C 64 nicht mehr benutzt werden. Der Selbsttest läuft wie folgt ab: Zunächst haben Sie etwa fünf Sekunden Zeit, um die Schreibschutzlichtschranke zu testen. Die rote LED am Laufwerk gibt darüber Auskunft, ob der Schacht lichtdurchlässig ist (Schreibschutz nicht aktiviert, rote LED ist an) oder ob ein Schreibschutzetikett auf der Diskette klebt (dann geht die rote LED aus). Nach fünf Sekunden läuft der Test automatisch weiter: Zunächst blinkt die rote LED viermal (Periodendauer: etwa eine Sekunde) zusammen mit dem Motor, der periodisch an- und ausgeht. Danach wiederholt sich dies wieder viermal, diesmal allerdings wechselseitig. Daraufhin werden der Hauptmotor und die Leuchtdiode eingeschaltet, und der Steppermotor getestet: Der Schreib/Lesekopf fährt zunächst 40 Tracks nach außen (auf Track 1), was beim ersten Mal ein Rattern zur Folge haben kann. Danach wird der Kopf 40 Tracks nach innen bewegt. Dieser Vorgang wiederholt sich dreimal. Danach beginnt der Selbsttest von vorn mit dem Test der Lichtschranke.

Dieser Test kann aber auch beim Einschalten des Laufwerks automatisch durchgeführt werden. Er startet unabhängig vom C 64, wenn beim Einschalten das Laufwerk auf Geräteadresse 10 oder 11 eingestellt ist, wenn also der Jumper 2 auf der Floppyplatine geöffnet ist. Hier könnten Sie ggf. einen Taster (Öffner) anschließen, den Sie vor und während des Einschaltens der 1541 drücken. So kann der sonst endlos dauernde Test auch unterbrochen werden: Wenn am Ende des Steppermotor-Tests dieser Taster gedrückt, also der Jumper 2 geöffnet ist, endet der Test und die Floppy führt die normale (schnelle) Reset-Routine durch. Nachteil: Sie ist danach auf die Adresse 10 eingestellt. Daher sollte das Drive gewöhnlich abgeschaltet werden, um den Test abzubrechen.

Neben dem praktischen Vorteil der schnelleren Resetroutine sowie der erhöhten IRQ-Frequenz erleichtert dieser problemlose Eingriff auch eventuelle Reparaturarbeiten am Laufwerk. Dieses kann, unabhängig vom C 64, geöffnet und untersucht werden. Zum Testen der fehleranfälligsten Komponenten verwenden Sie den Selbsttest. Die Fehler, die der von Commodore vorgesehene RAM/ROM-Test abfing, traten dagegen in der Praxis kaum auf.

Die Maschinensprache-Bibliothek

Warum das Rad neu erfinden? Wer in Maschinensprache programmiert, kann auf eine Vielzahl von Unterprogrammen zurückgreifen, die bereits fest im C 64 gespeichert sind. Wir verraten Ihnen, wie man bestimmte immer wiederkehrende Probleme in Assembler bewältigt. Diese Auflistung wird für Sie ein wertvolles Nachschlagewerk, wenn Sie in Maschinensprache programmieren. Wenn Sie schon über Grundkenntnisse in Assembler verfügen und Ihnen die notwendige Praxis vielleicht noch fehlt, so sind Sie hier genau richtig. Natürlich erheben wir hier keinen Anspruch auf Vollständigkeit. Allein mit der Auflistung aller wichtigen Routinen hätte man leicht ein ganzes Buch füllen können.

Aus Platzgründen können wir leider nicht auf alle Eigenarten genau eingehen.

Sollte Ihnen etwas unklar sein, ist es ein guter Tip, ein wenig zu experimentieren. Eine Erklärung wichtiger Begriffe finden Sie in dem ausführlichen Computer-Lexikon am Ende des Buches.

1. Mathematische Operationen

1.1. Addition von 16-Bit Zahlen

In den Speicherzellen 2 (Lowbyte) und 3 (Highbyte) befindet sich ein Wort, also eine 16-Bit Zahl. Zu dieser soll ein weiteres Wort addiert werden, das im gleichen Format in den Zellen 4 und 5 gespeichert ist. Das Ergebnis soll wieder in die Speicherzellen 2 und 3 geschrieben werden. Dazu verwenden wir den ADC-Befehl, der zum Inhalt des Akkumulators ein Byte und den Wert des Carry-Flags addiert. Bei einem Übertrag wird das Carry-Flag addiert.

```
LDA 2   ; Lowbyte 1 laden
CLC     ; Carry-Flag löschen
ADC 4   ; Lowbyte 2 addieren
STA 2   ; Ergebnis (Low) speichern
LDA 3   ; Highbyte 1 laden
ADC 5   ; dazu Highbyte 2 sowie evtl. Überlauf addieren
STA 3   ; Ergebnis speichern
RTS     ; fertig
```

1.2. Multiplikation

Der Befehl ASL (Arithmetical Shift left) rotiert die Bits des Wertes im Akkumulator um ein Bit nach links, wobei das Bit 0 auf Null gesetzt wird. Sie wissen vielleicht schon, daß das einer Multiplikation mit 2 gleichkommt. Um also den Wert im Akkumulator zu verdoppeln, geben Sie einfach den ASL-Befehl. Tritt dabei ein Überlauf auf, wird wiederum das Carry-Flag gesetzt. Um auch mit anderen Werten multiplizieren zu können, müssen wir den Vorgang in Multiplikationen mit Zweierpotenzen und Additionen zerlegen. Beispiel: Soll eine Zahl mit 10 multipliziert werden, müssen wir die Zahl erst mit vier multiplizieren, dann dazu den alten Wert der Zahl addieren und das Ergebnis wiederum verdoppeln. Probieren Sie's aus: Das entspricht genau der Multiplikation mit 10. In Assembler sieht eine Routine, die den Inhalt des Akkumulators mit 10 multipliziert, so aus (als Zwischenspeicher benutzen wir die sonst unbenutzte Speicherzelle 2):

```
STA 2   ; Wert speichern
ASL     ; Wert mal 2
ASL     ; Wert mal 2, zusammen also mal 4
ADC 2   ; dazu Original-Wert addieren
ASL     ; Ergebnis verdoppeln
RTS     ; fertig
```

Sollen dabei Überläufe berücksichtigt werden, die auftreten, wenn die zu multiplizierende Zahl größer als 25 ist, müssen wir nur nach jeder Operation das Carry-Flag testen. Ist es gesetzt, ist ein Überlauf eingetreten. In unserem Beispiel soll bei einem Überlauf die System-Routine zur Ausgabe eines ?OVERFLOW ERRORs (Adresse: hex. $B97E, dezimal 47486) aufgerufen werden.

```
    STA 2     ; Wert speichern
    ASL       ; Wert mal 2
    BCS ERR   ; bei Fehler
    ASL       ; Wert mal 2, zusammen also mal 4
```

```
    BCS ERR   ; bei Fehler
    ADC 2     ; dazu Original-Wert addieren
    BCS ERR   ; bei Fehler
    ASL       ; Ergebnis verdoppeln
    BCS ERR   ; bei Fehler
    RTS       ; fertig
ERR JMP 47486 ; Overflow Error
```

2. Ein-, Ausgabe

2.1. Zeichen am Bildschirm ausgeben

Die Routine BSOUT des Kernals wird in praktisch jedem Maschinenprogramm verwendet. Sie druckt wie der PRINT-Befehl einfach das Zeichen, dessen Ascii-Code im Akkumulator steht. BSOUT hat die Adresse 65490 (hex. $FFD2). Ein Beispiel: Folgendes Programm gibt das Wort »HALLO« aus, und beginnt danach eine neue Bildschirmzeile:

```
LDA #72   ; Code für »H«
JSR $FFD2 ; BSOUT, Zeichen ausgeben
LDA #65   ; Code für »A«
JSR $FFD2 ; BSOUT, Zeichen ausgeben
LDA #76   ; Code für »L«
JSR $FFD2 ; BSOUT, Zeichen ausgeben
JSR $FFD2 ; nochmals ausgeben
LDA #79   ; Code für »O«
JSR $FFD2 ; BSOUT, Zeichen ausgeben
LDA #13   ; Code für »CR«
JSR $FFD2 ; BSOUT, Zeichen ausgeben
RTS       ; fertig
```

Eine Tabelle mit den Ascii-Codes finden Sie im Anhang des Handbuchs. Die letzten Befehle kann man sogar noch weiter vereinfachen. Was viele Programmierer nicht wissen: Es gibt im System eine Routine, die ein CR (Carriage Return, beginne neue Bildschirmzeile, wirkt etwa wie die RETURN-Taste) ausgibt. Diese Routine hat die Adresse $AAD7 (43735). Statt der Befehle

```
LDA #13   ; Code für »CR«
JSR $FFD2 ; BSOUT, Zeichen ausgeben
```

könnte man also auch nur schreiben:

```
JSR $AAD7 ; CR ausgeben
```

Damit haben wir zwei Bytes gespart, die für den Befehl LDA #13 notwendig gewesen wären. Sie sehen schon, wie nützlich es ist, die Funktionsweise der wichtigsten Systemroutinen zu kennen.

2.2. Zahl ausgeben

Wir wollen die 16 Bit-Zahl, die im Beispiel 1.1. in den Speicherzellen 2 und 3 gespeichert wurde, numerisch am Bildschirm ausgeben. Dazu machen wir von der Betriebssystem-Routine AXOUT Gebrauch, die ein Wort ausgibt, dessen Lowbyte im X-Register und Highbyte im Akkumulator steht, ausgibt. Die Routine hat die Startadresse $BDCD (48589).

```
LDX 2     ; Lowbyte laden
```

```
LDA 3     ; Highbyte laden
JMP $BDCD ; AXOUT, Zahl ausgeben
```

Beispiel: Stand in Zelle 2 der Wert 123 und in Zelle 3 eine 8, so gibt das Programm die Zahl 2171 aus. Wie Sie mit oben angegebener Formel leicht nachrechnen können, steht LOW=123 und HIGH=8 für das Wort 2171. Beachten Sie, daß sich AXOUT nur zur Ausgabe von positiven ganzen Zahlen von 0 bis 65535 (Wertebereich eines Wortes) eignet.

2.3. Hexadezimalausgabe

Während die eben besprochene Routine AXOUT Zahlen immer dezimal ausgibt, ist manchmal auch die hexadezimale Ausgabe erwünscht. Die einfachste und kürzeste Lösung zeigen wir hier:

```
HEX      PHA        ; Zahl merken
         LSR        ; geteilt durch 2
         LSR        ; geteilt durch 4
         LSR        ; geteilt durch 8
         LSR        ; geteilt durch 16
         JSR HEX1   ; High-Nibble ausgeben
         PLA        ; Zahl zurückholen
HEX1     AND #15    ; Low-Nibble isolieren
         TAX        ; als Zeiger merken
         LDA HEX2,X ; Ziffer laden
         JMP $FFD2  ; und drucken (BSOUT)
HEX2     .ASC "0123456789ABCDEF"
```

Der Befehl LSR ist das Gegenstück zum oben schon erklärten ASL: Es teilt den Inhalt des Akkumulators durch zwei. Hier wenden wir diesen Befehl viermal an, um das High-Nibble (praktisch die Sechzehner-Stelle der Hexzahl) an die Bitpositionen 0 bis 3 zu schieben. Genau diese vier Bits werden von der Unterroutine HEX1 gedruckt. Die Bezeichnungen sind ab Position HEX2 im Ascii-Code gespeichert. Vielleicht sind Ihnen auch die Befehle PHA und PLA noch unbekannt. Sie bedienen den Prozessor-Stack: PHA speichert den Inhalt des Akkumulators auf dem Stack. Der Stack ist vergleichbar mit einem Papierstapel. PHA schreibt den Inhalt des Akkumulators auf ein Stück Papier und legt dieses oben auf den Stapel. PLA holt sich das oberste Blatt, liest den Inhalt und weist dem Akkumulator diesen Inhalt zu.

Die Routine HEX eignet sich für ein Byte (0 bis 255). Wird HEX mit einer 123 im Akkumulator aufgerufen, erscheint der hexadezimale Wert von 123 am Bildschirm: Ein 7B. Leicht läßt sich das Programm aber auch für größere Zahlen in mehreren Bytes anwenden: Wir wollen beispielsweise wieder das in den Adressen 2 (Low) und 3 (High) gespeicherte Wort hexadezimal ausgeben. Dabei ist zu beachten, daß zuerst das Highbyte angezeigt wird. Hexadezimalzahlen werden üblicherweise mit einem Dollarzeichen markiert, also geben wir auch dieses noch aus.

```
LDA #36    ; Dollarzeichen
JSR $FFD2  ; BSOUT, Dollar drucken
LDA 3      ; Highbyte laden
JSR HEX    ; und drucken
LDA 2      ; Lowbyte laden
JMP HEX    ; drucken und fertig
```

Befand sich etwa das Wort 32343 in den Speicherzellen 2 und 3 (Low: 87, High: 126), so druckt diese Routine völlig korrekt den Wert $7E57.

2.4. Text ausgeben

Einen längeren Text kann man im Prinzip mit einer Schleife Zeichen für Zeichen auslesen und mit BSOUT ausgeben. Aber es geht einfacher: Im System ist bereits eine Textausgabe-Routine vorhanden. Sie heißt STROUT und hat die Startadresse $AB1E (43806). Übergeben Sie dieser Routine im Akkumulator das Lowbyte, im Y-Register das Highbyte eines Textes, der irgendwo im Speicher im Ascii-Code steht. Das Ende des Textes muß durch ein Nullbyte markiert sein. Da STROUT im Gegensatz zum normalen Basic-Printbefehl am Ende des Textes nicht automatisch eine neue Zeile beginnt, dürfen Sie, falls das erwünscht ist, nicht vergessen, auch ein CR in den Text aufzunehmen. So sieht die Anwendung aus:

```
     LDA #<TEXT   ; Lowbyte laden
     LDY #>TEXT   ; Highbyte laden
     JMP $AB1E    ; STROUT Text drucken
TEXT .ASC "ICH GRUESSE DICH!"
                  ; Text im Ascii-Code
     .BYT 13,0    ; CR, Nullbyte als Endezeichen
```

Dieses Beispiel wirkt wie der Basicbefehl

```
PRINT "ICH GRUESSE DICH!"+CHR$(13);
```

2.5. Peripheriegeräte

Zum Thema Ein-/Ausgabe auf Peripheriegeräten, etwa Drucker oder Floppy, sei auf den Assemblerdateikurs verwiesen, den Sie ebenfalls in diesem Buch finden.

3. KAPITEL: GRAFIKPROGRAMMIERUNG

Sprites im Griff mit »Spritelist V2«

Sprites sind so praktisch! Wenn man doch nur Werkzeug hätte, mit dem man sie auch vernünftig verwalten könnte. »Spritelist« hilft Ihnen nicht nur beim »Klauen« von Sprites aus fremden Programmen, beispielsweise Spielen, sondern bietet darüberhinaus auch noch wertvolle und starke Funktionen zum Nachbearbeiten der Kobolde. Na, ist das nicht Banane?

Dieses Grafik-Utility dient in erster Linie dazu, Sprites aus eigenen oder fremden Programmen herauszuoperieren, um sie dann in eigenen Produktionen weiterverwenden zu können. Es hat also eigentlich nichts mit einem Spriteeditor zu tun, Funktionen zum Malen von Sprites sind nicht enthalten. Neben Möglichkeiten, ein noch im Speicher stehendes Programm nach Sprites abzusuchen, sind jedoch einige äußerst nützliche Funktionen eingebaut, die die Nachbearbeitung von gefundenen Sprites ermöglichen. Insoforn stellt »Spritelist« eine ideale Ergänzung zu einem Sprite-Editor dar. Es ist

vollkommen in stark optimierter Maschinensprache verfaßt und arbeitet dementsprechend schnell, sicher und flexibel. Dennoch brauchen Sie zur Anwendung keine Assemblerkenntnisse.

Um Sprites aus einem fremden Programm herauszuholen, sollten Sie dieses zunächst ganz normal laden und starten. Wenn die interessanten Sprites im Speicher sind, oder auf dem Bildschirm sichtbar werden, steigen Sie aus dem fremden Programm aus. Sollte dies nicht mit <RUN STOP/RESTORE> möglich sein, müssen Sie einen Reset-Taster verwenden. Ist das Programm Reset-geschützt, hilft nur noch ein erweitertes Betriebssystem, mit dem Sie diesen Schutz umgehen können (zum Beispiel das NSS-Kernal), oder der spezielle Reset-Taster aus diesem Buch.

Laden Sie dann unser Programm wie ein Basicprogramm mit dem Befehl

```
LOAD "SPRITELIST V2",8,0
```

und starten es mit RUN, da LIST keinen Programmtext zum Vorschein bringt. Das Hauptmenü erscheint auf dem Bildschirm. Unten finden Sie ein Feld, in dem das aktuelle Sprites zu sehen ist, und zwar links und der HiRes-Darstellung und rechts als Multicolor-Sprite. Darüber steht die Spritenummer (von 0 bis 1023 in Schritten von 64 Bytes), die Adresse, an der das Sprite im Speicher steht und die VIC-Bank, der das Sprite entstammt. Der im C64 eingebaute Videochip kann vier Banks ansprechen, das sind die Speicherbereiche, die er momentan adressieren kann. Bank 0 (Standard) geht von Adresse 0 bis 16383 (Sprites 0 bis 255), Bank 1 von Adresse 16384 bis 32767 (Sprite 256 bis 511), Bank 2 von 32768 bis 49151 (Sprites 512 bis 767) und Bank 3 umfaßt den Bereich zwischen 49152 und 65535 (Sprites 768 bis 1023).

Wählen Sie durch Druck auf die invers dargestellten Tasten die entsprechenden Funktionen, die im Folgenden in der Reihenfolge vorgestellt werden, in der sie auch auf dem Schirm zu sehen sind.

<I> - Invert: Das in der Anzeige sichtbare Sprite wird invertiert. Dieser Befehl wird durch nochmaliges Invertieren rückgängig gemacht.

<CLR> - Clear: Das in der Anzeige sichtbare Sprite wird gelöscht.

<E> - Expand: Der Vergrößerungsmodus der Anzeige wird ein- oder ausgeschaltet. Ist dieser Modus eingeschaltet, werden die beiden Sprites am Bildschirm waagerecht und senkrecht doppelt so groß gezeigt, neben dem Menüpunkt »Expand« blinkt ein Viereck.

<L> - Load: Ein Sprite kann von Diskette geladen werden. Dazu geben Sie den kompletten Filenamen ein und drücken <RETURN>. Drücken Sie diese Taste nur, ohne etwas einzugeben, wird die Funktion abgebrochen. Der Computer versucht nun, ein Sprite unter diesem Namen in den Speicher zu laden. Das Sprite wird immer in den Bereich ab 704 (Arbeitsbereich) geladen, das Programm schaltet nach dem Laden auf diesen Bereich um. Das Sprite sollte auf Diskette als PRG-File vorhanden sein, die ersten beiden Bytes des Files geben die Startadresse (beliebig) an, die folgenden 63 Bytes die Spritedaten im üblichen Format.

<S> - Save: Ein Sprite kann natürlich auch auf Diskette gespeichert werden. Dazu geben Sie den kompletten Filenamen ein und drücken <RETURN>. Drücken Sie diese Taste nur, ohne etwas einzugeben, wird die Funktion abgebrochen. Der Computer versucht nun, das in der Anzeige sichtbare Sprite unter diesem

Namen auf die Floppy zu speichern. Dabei wird ein PRG-File (Länge: 1 Block) erzeugt, die ersten beiden Bytes des Files geben die Startadresse (704=$2C0) an, die folgenden 63 Bytes die Spritedaten im üblichen Format.

<1 bis 4> - Farbe 1 bis 4: Durch Druck auf eine dieser Tasten wird die entsprechende Farbe um eins erhöht. Farbe 1 ist die Farbe des HiRes-Sprites, die Multicolor-Darstellung besteht aus den Farben 2 mit 4.

<Q> - Ende: Durch Druck auf diese Taste beenden Sie das Programm. Es kann, so es sich noch im Speicher befindet, mit SYS 32768 wieder gestartet werden.

<R> - Repeat: Mit dieser Taste wird die Wiederholfunktion der Tastatur ein- oder ausgeschaltet. Ist der Repeat eingeschaltet (zu erkennen an dem kleinen blinkenden Viereck im Hauptmenü), wirkt das dauerhafte Drücken einer Taste so, als ob Sie die Taste schnell hintereinander drücken und wieder loslassen.

<F1> - Plus 1: Die Adresse des Sprites, das in der Anzeige erscheint, wird um 64 Bytes erhöht, es wird also um ein Sprite weitergeblättert.

<F2> - Plus 10: Die Adresse des Sprites, das in der Anzeige erscheint, wird um 640 Bytes erhöht, es wird also um zehn Sprites weitergeblättert.

<F7> - Plus 100: Die Adresse des Sprites, das in der Anzeige erscheint, wird um 6400 Bytes erhöht, es wird also um 100 Sprites weitergeblättert.

<F3> - Minus 1: Die Adresse des Sprites, das in der Anzeige erscheint, wird um 64 Bytes erniedrigt, es wird also um ein Sprite zurückgeblättert.

<F4> - Minus 10: Die Adresse des Sprites, das in der Anzeige erscheint, wird um 640 Bytes erniedrigt, es wird also um zehn Sprites zurückgeblättert.

<F8> - Minus 100: Die Adresse des Sprites, das in der Anzeige erscheint, wird um 6400 Bytes erniedrigt, es wird also um 100 Sprites zurückgeblättert.

<F5> - Adr+1: Die Adresse des Sprites, das in der Anzeige erscheint, wird um eins erhöht. So lassen sich auch Sprites finden, die im Speicher nicht an einer durch 64 teilbaren Adresse beginnen.

<F6> - Adr-1: Die Adresse des Sprites, das in der Anzeige erscheint, wird um eins erniedrigt.

<*> - Restart: Das Programm »Spritelist V2« wird neu gestartet. Dabei schaltet der Computer alle Optionen ab und stellt Sprite Nr. 0 dar. Das Programm wird initialisiert.

<D> - Daten drucken: Es erscheint die Meldung, den Drucker vorzubereiten. Drücken Sie die Taste mit dem Pfeil nach links, wird diese Funktion abgebrochen. Nach Betätigung einer anderen Taste druckt ein abgeschlossener und eingeschalteter Drucker die 63 Sprite-Daten aus. Sie werden in vier Zeilen als rechtsbündige Dezimalzahlen ausgegeben. Diese Angaben könnte man direkt als Data-Zeilen in ein Basicprogramm übernehmen. Die Druckroutine ist für Drucker aller Art geeignet.

<P> - Sprite drucken: Es besteht die Möglichkeit, das Sprite selbst auszudrucken. Dazu wählen Sie zunächst, ob das Sprite in Originalgröße oder horizontal und vertikal in achtfacher Vergrößerung gedruckt werden soll. Die Ausgabe des nicht vergrößerten Sprites erfolgt im Grafikmodus, die Routine

wurde für MPS-Siebennadel-Drucker vom Typ MPS 801, 803 geschrieben. Drücken Sie die Taste <1>, <8> oder <Pfeil nach links>, je nachdem, ob die Originalgröße, die Vergrößerung oder doch kein Druck gewünscht wird. Dann erscheint ggf. die Meldung, den Drucker vorzubereiten. Drücken Sie die Taste mit dem Pfeil nach links, wird diese Funktion abgebrochen. Nach Betätigung einer anderen Taste druckt ein abgeschlossener und eingeschalteter Drucker das Sprite in der gewünschten Vergrößerung aus.

<X> - Spiegeln X: Das Sprite wird in der Waagerechten gespiegelt. Vor dem Spiegeln müssen Sie noch die Frage beantworten, ob es sich bei dem Sprite um ein Multicolor-Sprite handelt oder nicht. Mit der Pfeil-nach-links-Taste wird der Vorgang abgebrochen. Die Spiegelung kann durch ein erneutes Spiegeln rückgängig gemacht werden.

<Y> - Spiegeln Y: Das Sprite wird senkrecht gespiegelt. Die Spiegelung kann durch ein erneutes Spiegeln rückgängig gemacht werden. Hier spielt es keine Rolle, ob es sich um ein Multicolor-Sprite handelt oder nicht.

<F> - Rahmen: Um das Sprite läßt sich ein beliebig dicker Rahmen ziehen. Nach <F> geben Sie ein, welche Stärke der Rahmen, der alle vier Seiten umfaßt, haben soll. Zur Auswahl stehen 1 Punkt bis 8 Punkte (Tasten <1> bis <8>). Auch hier beendet die Taste mit dem Pfeil nach links die Funktion vorher. Wollen Sie keinen Rahmen aus gesetzten Punkten, sondern einen aus gelöschten Punkten, sollten Sie das Sprite erst invertieren (<I>), dann den Rahmen zeichen lassen und danach erneut invertieren.

<Pfeil nach links> - Suchen: Mit dieser Funktion kann ein Sprite im Speicher gesucht werden. Die Sprites in der Anzeige werden in einer bestimmten Geschwindigkeit durchgeschaltet, die Sie mit den Tasten <+> und <-> in den Grenzen 1 (sehr langsam) bis 32 (sehr schnell) ändern. Mit der Taste <R> wird die Richtung umgeschaltet. Blinkt neben diesem Untermenüpunkt ein kleines Viereck, blättert das Programm rückwärts, sonst vorwärts. Haben Sie beim Blättern das Sprite kurzzeitig gesehen, das Sie interessiert, war aber die Reaktion zu langsam, können Sie auf die Taste mit dem Pfeil nach links drücken. Der Computer blättert dann augenblicklich 12 Sprites in die andere Richtung. Je öfter Sie auf diese Taste drücken, desto schneller blättert das Programm in die Gegenrichtung der mit <R> gewählten Richtung. Die Taste <RUNSTOP> beendet diese Funktion und führt in das Hauptmenü zurück, das zuletzt angewählte Sprite bleibt in der Anzeige stehen und kann nun ggf. bearbeitet werden.

Zum Verständnis der letzten drei Funktionen ist es notwendig, einen kleinen Blick hinter die Kulissen des Spritefinders zu werfen. Obwohl in der Anzeige bis zu 1024 Sprites gezeigt werden können, handelt es sich doch immer um das selbe Sprite im Speicher, das der Computer darstellt. Es wird nämlich nicht etwa der Sprite-Zeiger 2040 auf den Speicherbereich gestellt, der dargestellt werden soll (das wäre ja auch nur auf eine Bank begrenzt und nicht bankübergreifend möglich), sondern es wird immer nur das Sprite im Speicher ab 704 = $2C0 (Bank 0, Sprite# 11) gezeigt. Das Programm kopiert den Speicherbereich, ab dem das Sprite eigentlich zu finden ist, jedes Mal nach 704, wenn Sie zum Beispiel mit den Funktionstasten oder der Suchfunktion auf ein anderes Sprite umschalten.

<A> - Auto: Die Kopierung des Bereiches, in dem das Sprite eigentlich steht, nach 704 geschieht gewöhnlich nur bei Bedarf, also zum Beispiel beim Umschalten auf ein anderes Sprite mit der Suchfunktion oder den Funktionstasten. Sie können jedoch auch eine Automatik einschalten, bei der das Kopieren unabhängig von gewählten Befehlen 60 mal in der Sekunden

IRQ-gesteuert abläuft. Die Automatik wird im Hauptmenü mit <A> ein- oder ausgeschaltet. Im eingeschalteten Zustand blinkt ein Viereck neben dem Menüpunkt. Sie erkennen den Unterschied leicht, wenn Sie das Sprite# 2 (Bank 0, Adresse 128) wählen und etwa auf die Mitte des Sprites achten. Ist die Automatik abgeschaltet, ändert sich das Aussehen des Sprites nicht. Bei eingeschalteter Automatik ist jedoch ein Flimmern in der Mitte des Sprites zu beobachten, das daran liegt, daß in diesem Spritebereich drei Speicherzellen enthalten sind, die vom System laufend verändert werden (die Systemuhr, Speicherzellen 160 mit 162). Zu beachten wäre noch, daß bei aktivierter Automatik die Modifizierungsbefehle (Spiegeln, Rahmen, Invertieren, Löschen) unwirksam sind - außer Sie bearbeiten das Sprite Nr. 11. Denn diese Befehle wirken im Speicher auf den Bereich ab 704, der dargestellt wird, nicht auf den Original-Bereich. Wird etwa das Sprite# 123 dargestellt, ist der Originalbereich der ab 123*64 = 7872 = $1EC0, die Änderungen wirken aber ab 704 = $2C0. Im Normalbetrieb des Programmes sollte die Automatik abgeschaltet sein.

<Z> - Zurück: Wenn Sie diesen Befehl wählen, wird der Bereich ab 704 in den Originalbereich kopiert. Beispiel: Sie bearbeite das Sprite# 123, das im Speicher (siehe oben) ab 7872 steht. Es wird nach 704 kopiert, sobald Sie das Sprite# 123 einstellen. Jetzt kann es zum Beispiel invertiert werden (Taste <I>). Das invertierte Muster findet sich jetzt ab 704, und kann zum Beispiel gespeichert werden (beim Speichern wird immer der Bereich ab 704 gespeichert). Im Original-Bereich ab 7872 jedoch befindet sich immer noch das nicht invertierte Ogirinal-Sprite, und zwar so lange, bis Sie <Z> betätigen. Achten Sie darauf, daß Sie diesen Befehl nicht bei Sprites mit einer Nummer kleiner als 11 anwenden, da sonst Sprite-Daten in die Zeropage kopiert werden und - falls Sie das Sprite verändert haben - ein Absturz möglich ist. Ähnliche Effekte sind zu beobachten, wenn Sie diesen Befehl auf eines der Sprites 16, 17, 18 oder 19 anwenden (Bildschirm) oder ein Sprite mit einer Nummer zwischen 512 und ungefähr 564 anwenden (in diesem Bereich liegt das Hauptprogramm des Sprite-Finders).

<U> - Undo: Wenn Sie diesen Befehl wählen, wird der Originalbereich erneut in den Bereich ab 704 kopiert. So läßt sich eine Undo-Funktion realisieren. Beispiel: Sie bearbeiten wieder das Sprite# 123, das im Speicher ab 7872 steht. Jetzt legen Sie einen Rahmen der Dicke 3 um das Sprite. Nach dieser Manipulation gefällt Ihnen das Sprite jedoch nicht mehr. Da im Originalbereich ab 7872 noch das unveränderte Original-Sprite zu finden ist, können Sie es sich durch Druck auf <U> wieder in die Anzeige zurückholen.

Das war eine Beschreibung aller Programmfunktionen des Spritefinders. Drücken Sie eine unerlaubte Taste, ertönt ein Signal. Wir wünschen Ihnen viel Spaß und gelungene Sprites mit diesem Tool! Beachten Sie aber bitte, daß es nicht nur in Deutschland verboten und strafbar ist, Teile fremder Programme, also zum Beispiel Sprites, in eigenen Programmen weiterzuverwenden und diese dann zu vertreiben (Urheberrechtschutz).

Anleitung zum »Chartransposer« Version 1.0

Dieses Programm dient in erster Linie zum Nachbearbeiten, Verschieben und Aufbereiten von fertigen Zeichensätzen. Viele Sonderfunktionen machen dieses

Utility vor allem für Grafikprogrammierer interessant.

Laden Sie das in reiner Maschinensprache geschriebene Programm mit

LOAD"CHAR*",8

und starten mit RUN. Die Bildschirmmaske erscheint. Oben stehen die verfügbaren Kommandos, die Tasten, die gedrückt werden müssen, sind jeweils durch Großbuchstaben markiert. Unten ist der momentan in Arbeit befindliche Zeichensatz eingeblendet. Drücken Sie die Taste <O>, um ein vernünftiges Bild zu erhalten.

Die Funktionen im einzelnen:

Invert (Taste <I>):

Diese Funktion invertiert den gesamten Zeichensatz, bzw. den eingestellten Bereich (siehe unten). Durch nochmaliges Invertieren wird der Effekt wieder rückgängig gemacht.

Load (Taste <L>):

Mit dieser Funktion kann ein Charset geladen werden. Die Startadresse des Char-Files (Länge: maximal 9 Blocks, Byte 1 und 2 = Startadresse, dann Zeichendaten im üblichen Format) auf der Diskette wird ignoriert, der Satz wird immer an die selbe Stelle im Speicher geladen. Während Diskettenoperationen wird wegen des Raster-IRQs der Bildschirm flimmern, was jedoch zu keinerlei Fehlfunktionen führt.

Save (Taste <S>):

Hiermit wird ein im Speicher befindlicher Zeichensatz als neun Blocks langes File auf Diskette gespeichert. Obwohl das Utility die Zeichensätze immer an der selben Stelle im Speicher verwaltet, wird dem erzeugten File auf Diskette die im Programm vorgegebene Startadresse (wird ganz unten am Bildschirm eingeblendet und kann mit der Taste <W> verändert werden) gegeben. Das Umkopieren von Zeichensätzen ist auch eines der Haupt-Anwendungsgebiete von »Chartransposer« (zum Beispiel, wenn Sie aus einem Programm ein Charset »geklaut« haben, dies aber auf der Diskette noch die falsche Adresse hat und Sie keinen Diskmonitor bemühen wollen): Laden Sie den Zeichensatz mit der LOAD-Funktion, ändern Sie die Startadresse wie gewünscht, modifizieren Sie ggf. den Satz und speichern Sie ihn dann mit der neuen Adresse wieder ab.

Unabhängig vom eingestellten Bereich (siehe unten) wird immer der gesamte Zeichensatz gespeichert.

Quit (Taste <Q>):

Diese Funktion verläßt das Programm. Es kann, sofern es sich noch im Speicher befindet, mit SYS 2084,1 oder aber mit SYS 32768 wieder gestartet werden. Gespeicherte Zeichensätze gehen nicht verloren.

Mirror X (Taste <X>):

Mit dieser Funktion kann der gesamte Zeichensatz oder der eingestellte Bereich (siehe unten) waagerecht gespiegelt werden: aus dem Buchstaben »b« wird ein »d«.

Mirror Y (Taste <Y>):

Mit dieser Funktion kann der gesamte Zeichensatz oder der eingestellte Bereich (siehe unten) senkrecht gespiegelt werden: aus dem Buchstaben »M« wird ein »W«.

Inv. Half (Taste <V>):

Haben Sie einen Zeichensatz, der zwar in der oberen Hälfte (nicht-reverse Zeichen) ganz gut aussieht, aber in der unteren Hälfte, in der normalerweise die inversen Zeichen stehen, nicht vollständig, zerstört oder mit für Sie unnötigen Sonderzeichen versehen ist, berechnen Sie mit Hilfe dieser Funktion, die sich auch immer auf den gesamten Zeichensatz bezieht, die untere Hälfte. Arbeitsweise: Die 128 Zeichen der oberen Hälfte werden invertiert und in die untere Hälfte kopiert.

Test (Taste <T>):

Mit dieser Funktion testen Sie die Wirkung eines fertigen Zeichensatzes auf dem Bildschirm. Der Schirm wird gelöscht, der neue Zeichensatz eingeschaltet. Jetzt können Sie den ganz normalen Bildschirmeditor benutzen. Verlassen wird diese Funktion mit der Taste <f1>.

Lft. Margin (Taste <F>):

Es besteht die Möglichkeit, einige Befehle nur auf einen bestimmten Bereich des Zeichensatzes wirken zu lassen. In der Mitte des Schirms sehen Sie nach dem Start die Anzeige »left margin: 0; right margin: 255«. Dies bedeutet, daß sich diese Befehle momentan auf den gesamten Zeichensatz (von Zeichen 0 bis 255 einschließlich) beziehen. Drücken Sie die Taste <F>, erscheint ein weißer Rahmen, den Sie mit den Cursortasten nun nach links und rechts auf jedes beliebige Zeichen bewegen können (bis zum rechten Rand). Nach dem Druck auf <RETURN> wird der neue linke Rand gespeichert, die Befehle <I>, <X>, <Y>, <C>, <P>, <A> und <H> beziehen sich jetzt nur noch auf diesen Bereich.

Rgt. Margin (Taste <R>):

Ebenso wie der linke Rand kann auch die Position des rechten Randes frei verschoben werden. Die Bedienung erfolgt analog wie oben.

Rom Set (Taste <O>):

Soll mit den beiden eingebauten Commodore-ROM-Zeichensätzen gearbeitet werden, verwenden Sie diese Funktion. Dadurch wird der im Speicher befindliche Satz gelöscht und der CBM-Zeichensatz darüberkopiert. Hinter der Anzeige »Rom Set« erscheint die Ziffer 0, wenn momentan der Groß/Grafik-Zeichensatz aktiviert ist, die 1 weist auf Klein/Großschrift hin.

Clear (Taste <C>):

Dieses Kommando löscht alle Zeichen im eingestellten Bereich ersatzlos und unwiderruflich.

Prepare (Taste <P>):

Diese Funktion ist etwas ganz besonderes. Sie konvertiert einen Hires-Zeichensatz ins Multicolor-Format. Da hierbei jedoch die Auflösung auf die Hälfte verringert wird, sollten Sie nicht normale Hires-Zeichensätze damit behandeln, sondern nur extra dafür vorgegebene. Funktionsweise: Bei der Konvertierung wird zeilenweise vorgegangen. In jeder Zeile (Byte) finden sich acht Bit, von denen immer zwei nebeneinanderliegende Bits zu einer von vier Gruppen zusammengefaßt werden. Ist in einer Gruppe mindestens ein Bit gesetzt, werden beide Bits auf 1 gesetzt und dadurch beide Pixel gesetzt:

```
vorher:  nachher:
  0 0      0 0
  1 0      1 1
  0 1      1 1
  1 1      1 1
```

Work Area (Taste <W>):

Diese Funktion dient dazu, die in der untersten Bildschirmzeile angezeigte »work area« festzulegen. Dieser Speicherbereich hat für die Arbeit mit dem Programm »Chartransposer« keine Bedeutung, sie legt fest, welche Startadresse Zeichensätze beim Speichern mit der Save-Funktion erhalten sollen. Defaultwert ist $3800 (dezimal 14336), dies ist auch die Position, an der die Zeichensätze während der Bearbeitung im Speicher liegen - unabhängig von der jeweils eingestellten Work Area.

Rotate (Taste <A>):

Mit dieser Funktion werden die Zeichen innerhalb des gewählten Bereiches um 90° im Uhrzeigersinn gedreht. Viermaliges Drehen bringt wieder den Ausgangszustand, ebenso zweimaliges Drehen und dann Spiegeln in X- und Y-Richtung.

Restart (Taste <E>):

Dient zum Neustarten des »Chartransposers«. Dabei werden alle Einstellungen auf die Standardwerte gebracht, der Zeichensatz jedoch nicht gelöscht. Diese Funktion ist weiter ganz praktisch, wenn man ein Hardcopy-Programm zum Ausdrucken des Bildes benutzt hat, welches den Raster-IRQ abgeschaltet hat (z.B. Uniprint).

Catalog (Taste <G>):

zeigt das Directory der eingelegten Diskette durch waagerechtes Scrollen in der untersten Bildschirmzeile an. Durch Druck auf <SPACE> wird die Ausgabe angehalten, <STOP> bricht sie ab.

Disk Cmd (Taste <D>):

Mit dieser Funktion können Sie Befehle über den Kommandokanal 15 an das Laufwerk senden. Geben Sie die Befehlszeile ein und drücken <RETURN>. Daraufhin wird gleich der Fehlerkanal angezeigt. Wollen Sie nur die Statusmeldung lesen, drücken Sie nur <RETURN>, ohne einen Befehl einzugeben.

Info (Taste <N>):

Diese Funktion informiert Sie auf einer Tafel kurz über das Programm. Nach einem beliebigen Tastendruck geht's weiter.

Show Area (Taste <H>):

Diese Funktion färbt innerhalb der Zeichensatz-Anzeige den mit den Tasten <F> und <R> gewählten Ausschnitt weiß.

Das waren alle Funktionen des Programmes »Chartransposer«. Wir wünschen Ihnen viel Spaß und viele gelungene Zeichensätze damit!

Anleitung zu »Softscroll«

Dieses kurze Hilfsprogramm bindet eine Softscroll-Routine ins Betriebssystem ein. Wer sich schon immer daran gestört hat, daß das Scrollen zum Beispiel beim LIST-Befehl ziemlich ruckweise und auch etwas schnell vonstatten geht, wird sich über diese Routine freuen. Sie eignet sich aber auch vorzüglich dazu, in eigenen Programmen beispielsweise scrollende Vorspänne filmartig zu steuern.

Die Routine wird mit dem Befehl

```
LOAD "SOFTSCROLL",8,1
```

von Diskette geladen. Geben Sie danach NEW ein. Nach dem Start mit

```
SYS 49152
```

installiert sich das Programm jetzt im Speicher und wird dann automatisch gestartet. Sobald sich der Computer mit READY. zurückmeldet, ist die neue Scrollroutine aktiviert. Probieren Sie sie aus, indem Sie zum Beispiel eine lange Zahlenkolonne ausgeben:

```
FOR I = 1 TO 20000 : PRINT "DIESE ZAHL HEISST";I : NEXT
```

Bei Verwendung in einem Basicprogramm kann die Scroll-Routine beispielsweise mit folgender Zeile automatisch nachgeladen und aktiviert werden:

```
10 IF SC=0 THEN SC=1: LOAD "SOFTSCROLL",8,1
20 SYS 49152
```

Auch die Scroll-Geschwindigkeit kann ggf. variiert werden. Der Befehl

```
POKE 49263,X
```

stellt X (1 bis 255) als Geschwindigkeit ein. Die Voreinstellung ist 2, je kleiner X ist, desto schneller, aber auch ruckartiger scrollt der Bildschirm. Der Effekt kann außerdem mit

POKE 1,55

abgeschaltet werden, die Re-Aktivierung erfolgt dann mit

POKE 1,53

Nach einem Reset sollten Sie die Routine mit

SYS 49152

neu starten, falls sie sich noch im Speicher befindet. Sie belegt den Speicherbereich 49152 bis 49281.

Doppelt gemoppelt mit DoublePrint

Haben Sie etwas Wichtiges zu sagen? Dann machen Sie es doch auch auf dem Bildschirm kenntlich! Dieses Utility vergrößert alle Textausgaben auf das Doppelte.

Manchmal ist es ganz gut, wenn man die Bildschirmausgabe mit einem besonderen Zeichensatz interessanter machen kann. DoublePrint enthält etwas wirklich einzigartiges: Einen 20-Spalten Zeichensatz.

Warum sollte man von 40 Zeichen pro Zeile auf 20 Zeichen umschalten? Zum einen ist ein Wort um so leichter zu lesen, je größer es ist. Zum anderen ist es ein netter Trick, wenn Sie die Aufmerksamkeit des Anwenders erregen wollen (z.B. bei Werbeprogrammen, die im Schaufenster ablaufen).

Wenn sich das Tool DoublePrint im Speicher befindet, ist es ganz einfach, zwischen dem 20 und 40 Zeichenmodus umzuschalten, im Programm- oder Direktmodus. Laden Sie zunächst die Routine mit

LOAD "DOUBLE PRINT",8,8

und starten Sie sie mit

SYS 49152:NEW

Der NEW-Befehl, der durch einen Doppelpunkt von SYS-Befehl abgetrennt wird, ist sehr wichtig. Er löscht den Basicspeicher, der hier an eine andere Stelle verschoben wurde. Wenn Sie später in Basicprogrammen umschalten wollen, geben Sie nur ein:

SYS 49152

Geben Sie einmal

LIST

ein. Während Sie tippen, sehen Sie nur konfuse Zeichen. Drücken Sie

<RETURN>. Das Listing, sofern es vorhanden ist, wird ganz korrekt im 20 Zeichenmodus ausgegeben.

Der Grund für die seltsamen Zeichen ist beim C 64 technisch bedingt. Da der Computer eigentlich nichts anderes als 40 Zeichen pro Zeile darstellen kann, ist die 20 Zeichen-Darstellung gar nicht echt, sondern wird durch einen veränderten Zeichensatz erzeugt. Jedes doppelt breite Zeichen besteht aus zwei normalen Zeichen. DoublePrint hat dazu die Print-Routine des Betriebssystems verändert. Immer wenn der C 64 auf den Bildschirm schreibt (außer, wenn Sie etwas eingeben) wird jedes Zeichen zweimal ausgegeben, einmal normal und einmal revers. Der Zeichensatz wurde so umdefiniert, daß die normalen Zeichen die linke Hälfte der Zeichen enthalten und die reversen Zeichen die rechte Hälfte. Dies ist auch der Grund dafür, daß man bei aktiviertem 20 Zeichenmodus keine normal- oder doppelt breiten inversen Zeichen darstellen kann. Wenn die beiden Hälften von der CHROUT-Routine direkt nebeneinander gedruckt werden, entsteht ein doppelt breites Zeichen.

Versuchen Sie einmal folgendes: Geben Sie ein <A> ein. Nun schalten Sie den Reversmodus mit <CTRL 9> ein und drücken noch einmal <A>. Der komplette Buchstabe erscheint. Jetzt versuchen wir es mit einem Direktmodus-Kommando. Gehen Sie mit dem unsichtbaren Cursor (es gibt ja keine reversen Zeichen mehr) in eine leere Bildschirmzeile und geben ein:

PRINT "HALLO"

Wieder können Sie nicht lesen, was Sie da schreiben. Wenn Sie jetzt jedoch <RETURN> drücken, wird völlig korrekt HALLO breit auf dem Bildschirm ausgegeben.

Nun schalten Sie mit dem Befehl

SYS 49155

den normalen 40-Zeichenmodus wieder ein. Sie sehen jetzt ganz klar, auf welche Weise die neuen breiten Zeichen erzeugt werden.

Wollen Sie in einem eigenen Programm Eingaben im 20 Zeichen Modus vornehmen lassen, müssen Sie sich eine eigene Eingaberoutine schreiben, in der die Zeichen nicht mit INPUT, sondern mit GET eingelesen und dann zweimal ausgedruckt werden.

Das Maschinenprogramm, das DoublePrint steuert, liegt ab 49152 im Speicher. Der neue Zeichensatz wurde bei 2048 untergebracht, der Basicspeicher beginnt jetzt bei 4096.

Anleitung zum Programm »LETTER V2«

Dieses kurze Utility erlaubt die Darstellung von vergrößerten Buchstaben am Zeilenanfang, wie es in Zeitschriften als »Initials« manchmal am Anfang eines Absatzes verwendet wird.

Um sich von den Fähigkeiten zu überzeugen, sollten Sie sich zunächst einmal

das Demoprogramm ansehen. Laden Sie zunächst das Programm »Letter« mit den Befehlen

```
LOAD "LETTER V2 $C000",8,8
NEW
SYS 49152
```

Nun erscheint die Einschaltmeldung, die bereits vergrößerte Zeichen enthält. Drücken Sie nun die Tasten <RUN-STOP> und <RESTORE>, um die Sprites für das Erste wieder abzuschalten. Wenn Sie nun das Demoprogramm laden, kann es sonst nämlich passieren, daß der Computer abstürzt. Wegen des sehr empfindlichen Timings ist es beim C 64 bei Diskettenzugriffen nicht erlaubt, daß Sprites eingeschaltet sind.

Geben Sie nun ein

```
LOAD "LETTER V2 DEMO",8
RUN
```

Das Demo wird nun gestartet. Sie zeigt kurz die Möglichkeiten von »Letter V2«.

Wie wird das Programm bedient? Wie Sie vielleicht bereits bemerkt haben, erfolgt die Darstellung der »Initials«, die übrigens nur am Zeilenanfang erlaubt sind, vollautomatisch. Nach dem Programmstart mit SYS 49152 ist die Routine so eingestellt, daß jedesmal nach der Ausgabe von <RETURN> der nächste Buchstabe groß dargestellt wird. In dieser Betriebsart würde also jede Zeile mit einem Initial beginnen. Daher wurden noch vier Steuerzeichen eingebaut, die Sie in den auszugebenden Text einbauen können. Diese Codes wirken nicht, wenn sie direkt bei blinkendem Cursor über die Tastatur eingegeben werden, sondern werden erst dann wirksam, wenn der C 64 sie in einem String beispielsweise bei PRINT ausgibt.

Die Codes lauten:

<CTRL A> oder CHR$(1): Die Ausgabe der vergrößerten Buchstaben wird eingeschaltet. Ist der Spezialmodus (siehe unten) aktiviert, dient dieser Code dazu, die nächste folgende Zeile mit einem Initial zu versehen.
<CTRL B> oder CHR$(2): Die Ausgabe der Initials wird vollkommen abgeschaltet
<CTRL C> oder CHR$(3): Der Spezialmodus (siehe unten) wird abgeschaltet
<CTRL D> oder CHR$(4): Schaltet den Spezialmodus an. Hier wird an den Anfang einer Zeile nur dann ein großer Buchstabe gestellt, wenn in der Zeile davor ein <CTRL A> vorkam. Dieser Modus ist nach dem ersten Laden abgeschaltet, sollte aber eingeschaltet werden, da man mit ihm »Letter V2« am besten kontrollieren kann: Nur am Anfang eines neuen (inhaltlichen) Absatzes soll ein Initial stehen.

Ein Beispiel:

```
10 PRINT "ES FOLGT IN DER NAECHSTEN ZEILE EIN INITIAL:"CHR$(4)CHR$(1)
20 PRINT "DAS 'D' WIRD VERGROESSERT, NICHT JEDOCH DAS FOLGENDE 'A':
30 PRINT "AB JETZT WERDEN WIEDER ALLE "CHR$(3)CHR$(1)"ZEILENANFAENGE ALS
INITIAL DARGESTELLT."
40 END
```

Wichtig ist noch zu wissen, daß sich niemals mehr als acht Initials gleichzeitig auf dem Schirm befinden können, da die großen Buchstaben Sprites sind, von denen der C 64 hardwaremäßig nur acht darstellen kann. Wird diese Anzahl überschritten, wird die Ausgabe solange wieder normal

geschaltet, bis wieder Sprites frei sind.

Falls Sie »Letter V2« mit einem eigenen Zeichensatz verwenden wollen, teilen Sie der Erweiterung die Startadresse der neuen Zeichen mit dem Befehl

SYS 49152, A, B

A ist hierbei die Startadresse im Speicher, B ist der Wert von PEEK (1) während des Auslesens der Zeichen. Wenn Sie nur SYS 49152 geben, werden die Standardwerte gesetzt (nur, wenn Sie vorher nicht mit dem erweiterten SYS-Befehl geändert wurden). Diese sind:

SYS 49152, 53248, 51

Wie funktioniert »Letter V2« intern? Die Erweiterung besteht aus drei Teilen. Der erste Teil, die Initialisierung, läuft nur einmal bei SYS 49152 ab und »baut« das Programm in den C 64 ein. Der zweite Teil wird jedesmal bei der Ausgabe eines Zeichens aufgerufen (BSOUT), prüft zunächst, ob die Ausgabe auf den Bildschirm gehen soll und ob Steuerzeichen (<CTRL A> und so weiter) gesendet werden. Ansonsten wird geprüft, ob dem zu druckende Zeichen ein <RETURN> (Zeilenende) vorausging. Wenn dies so ist und die Ausgabe als Initial erlaubt und möglich ist, wird an der entsprechenden Stelle am Bildschirm ein reverses <CBM B> (Bildschirmcode 255) in der Hintergrundfarbe (somit ist es unsichtbar) gesetzt. Dieses dient als Erkennungszeichen für das Initial. Daher dürfen Sie bei aktivierter Erweiterung dieses Zeichen nicht am Zeilenanfang verwenden, diese kleine Einschränkung dürfte jedoch in der Praxis nicht schlimm sein. Ein Initial hat die Größe von vier normalen Blockgrafikzeichen, belegt also vier Bildschirmspeicherzellen. Links oben steht wie gesagt der Erkennungscode, rechts oben wird nun die Nummer des Sprites eingetragen, die beiden unteren Zeichen enthalten den Farbcode für das Initial. Nun wird das Zeichen in ein freies Sprite übertragen, bei Bedarf natürlich auch revers, und der restliche Text ausgegeben.

Der dritte Teil von »Letter V2« läuft im Interrupt ab. Hier wird jede 1/60 Sekunde geprüft, ob das Erkennungszeichen (255) irgendwo am Zeilenanfang steht, und wenn ja, das Sprite dorthingesetzt. Durch diese entwas komplizierte Technik wird das Scrollen der Initials ermöglicht.

Hier nun noch die Speicherbelegung des Programms (nur Version 2.0, hexadezimal):

```
0002-0005 temporär
0800-09FF Spritebereich
0A00-9FFF Basic-Programm
C000-C302 »Letter V2«
C000-C005 Sprungtabelle
C006      Arbeitskopie von 53269 (Sprite-Enable)
C007      Bildschirmzeile für Initial
C008-C009 Basisadresse Zeichensatz (A bei SYS 49152,A,B)
C00A      Konfiguration Zeichensatz (Parameter B)
C00B      BSOUT Zwischenspeicher Y Register
C00C      BSOUT Zwischenspeicher X Register
C00D      BSOUT Zwischenspeicher Akku
C00E      Flag: letztes Zeichen war <RETURN>
C00F      Aktuelle Farbe für Initial
C010      Flag: Initial revers darstellen
C011      Nummer des Sprites für nächstes Initial
C012      Y-Registerstand zum Lesen aus Zeichenspeicher
```

C013   Y-Registerstand zum Schreiben in Initial
C014   relative Adresse des Zeichens high (Groß/Klein od. Grafik)
C015   Flag: Initial erlaubt (0=ja)
C016   Flag: Spezialmodus (255=ja)
C017-C01E Zweierpotenzen
C01F-C07B Einschaltmeldung
C07C-C302 100% Maschinenprogramm
C07C   Init-Routine
C11B   neue IRQ-Routine
C127   Initials suchen und ausgeben
C189   Schaltroutinen für Steuerzeichen
C1A6   neue BSOUT-Routine
C21F   Initial ausgeben
C27C   Zeichen in Sprite kopieren
C302   letztes Byte

Formatierte Zahlenausgabe mit »PRINT USING«

Eine »Print-Using-Routine« ist zwar grundsätzlich nichts neues. Erfahrungsgemäß hat man aber nie eine zur Hand, wenn man eine braucht. Wir präsentieren Ihnen daher eine neue besonders komfortable und leistungsstarke Version dieses kurzen Unterprogramms zur formatierten Ausgabe einer Zahl.

Was tun Sie, wenn Sie in Basic eine Zahl oder ein Rechenergebnis auf dem Bildschirm oder Drucker ausgeben möchten? Sie verwenden den PRINT-Befehl: Zum Beispiel einfach

PRINT X

Eigentlich funktioniert das ganz gut. Spätestens jedoch, wenn Sie formatierte Tabellen oder sonstige geordnete Zahlenaufstellungen ausgeben möchten, stellen Sie einen wesentlichen Mangel des PRINT-Befehles in Basic V2 fest: Er wirft »Kraut und Rüben« wirr durcheinander. Zahlen werden linksbündig ausgegeben, von Ordnung kann keine Rede sein. Erwarten Sie, daß alle Dezimalpunkte schön untereinander stehen, Einer unter Einer, Zehner unter Zehner, Nachkommastellen an der selben Stelle beginnend? Soll vielleicht, wie in Deutschland üblich, statt des Punktes ein Komma zur Abtrennung der Nachkommastellen erscheinen, und dafür der Punkt zur Abtrennung der Tausender? Schauen Sie einmal auf Ihren Bank-Kontoauszug: Da steht nicht DM 51351,3425, sondern ordentlich DM 51.351,34 (ich wünsche Ihnen einen so hohen positiven Kontostand!). Wünschen Sie das Vorzeichen hinter statt vor der Zahl? Soll auf eine bestimmte Anzahl Nachkommastellen gerundet werden? Stört Sie, daß bei Zahlen zwischen 0 und 1 die Null vor dem Punkt fehlt? Jaja, der PRINT-Befehl ist bei numerischen Ausgaben sehr schwach auf der Brust.

Professionelle Programme enthalten daher eine »PRINT USING« Routine, die die Zahl vor der Ausgabe in einen String wandelt, dort die notwendigen Änderungen vornimmt und dann den String ausgibt. Solche Routinen in Basic haben einige Nachteile: Sie sind schwer zu programmieren, meist nicht universell einsetzbar, kosten Zeit und - ganz wichtig - produzieren eifrig String-Müll, der über kurz oder lang zur gefürchteten Müllabfuhr »Garbage collection« im Computer führt. Und die kann bis zu einigen Minuten dauern.

Es liegt daher nahe, eine Konvertierungsroutine numerisch -> formatierten String in Maschinensprache zu formulieren. Die Routine, die wir Ihnen bieten, beseitigt nicht nur ALLE weiter oben genannten Nachteile. Sie ist darüberhinaus sehr komfortabel und einfach in der Anwendung. Sie brauchen lediglich grundlegende Basic-Kenntnisse zur Anwendung des Hilfsprogramms, das leicht in eigene Basicprogramme eingebaut werden kann.

Zum Test laden Sie die nur zwei Blocks kurze Routine mit

```
LOAD "USING",8,8
```

Danach geben Sie NEW ein. Die Routine wird mit folgendem, nur auf den ersten Blick kompliziert erscheinenden Befehl aktiviert:

```
SYS 51200,X,LE,VZ,TS,NK,FZ,CA,X$
```

Die Buchstaben hinter der Adresse 51200 stellen die Parameter dar, die die Funktionsweise und damit das Aussehen der gewandelten Zahl steuern. Sie können hier Zahlen einsetzen, Variablen oder numerische Basic-Terme. Sehr wichtig sind die acht Kommas, die die Parameter abtrennen:

X ist die numerische Variable, deren Wert ausgegeben werden soll, oder ein Rechenausdruck, oder auch eine Klartext-Zahl.

LE bestimmt die Länge des konvertierten Ausdrucks in Zeichen. Wird der Ausdruck länger als LE, erscheint der ?STRING TOO LONG ERROR. Ist der erzeugte Ausdruck zu kurz, wird er von links mit dem Füllzeichen FZ (siehe unten) aufgefüllt (man sagt »rechtsbündige Ausgabe«).

VZ bestimmt die Position des Vorzeichens. Ist VZ=0, so wird das Vorzeichen unterdrückt, es erscheint also nur der Absolutwert der Zahl. Ist VZ=1, so erscheint das Vorzeichen direkt vor der ersten Stelle der Zahl (Minuszeichen oder Leerzeichen für Zahlen über Null), so wie wir das von PRINT kennen. Ist VZ=128, so stellt der Computer das Vorzeichen hinter die Zahl.

TS nimmt entweder den Wert 0 oder 1 an. Ist TS=1, so trennt der Computer die Tausender und Millionen-Stellen durch einen Punkt ab. Eintausendvierhundert ist dann 1.400 statt 1400, fünfzehn Millionen dreihundert erscheint als 15.000.300 statt 15000300. Ist TS=0, finden sich keine Trennpunkte in der Ausgabe.

NK ist die Anzahl der Nachkommastellen und bewegt sich im Bereich von 0 (kein Nachkommaanteil) bis 9 (neun Nachkommastellen). Das Programm nimmt dabei automatisch die erforderliche 4/5-Rundung vor. Soll etwa 135.628 auf zwei Nachkommastellen genau ausgegeben werden, ist das Ergebnis aufgerundet 135.63. Wählen Sie NK=0, so erscheinen zwar keine Nachkommastellen, dafür aber ggf. der Punkt bzw. das Komma (siehe unten CA). 135.628 auf 0 Stellen genau liest sich mithin 136.

FZ ist der ASCII-Code des Füllzeichens, mit dem der erzeugte String von links aufgefüllt wird, bis er die gewünschte Länge (LE) erreicht hat. So werden Zahlen rechtsbündig, die einzelnen Stellen stehen brav untereinander. Gewöhnlich soll mit dem Leerzeichen aufgefüllt werden, FZ ist dann 32. Auch das »Scheckfälschungszeichen«, der Stern, kann Verwendung finden, dann setzen Sie für FZ die 42 ein. Jetzt kann kein böser Bube Ihre Eurocheques mehr »korrigieren«, wenn Sie sie mit dem C 64 und unserer Using-Routine bedrucken. Eine Tabelle dieser ASCII-Codes finden Sie im C 64-Handbuch im

Anhang.

CA ist entweder 0, 1, 2 oder 3 und entscheidet über das Trennzeichen zwischen Vor- und Nachkommastellen (man sagt »Dezimalseperator«). Ist CA=0, so wird der Nachkomma-Anteil unabhängig (!) vom Parameter NK total abgeschnitten. Setzen Sie also CA nur dann auf Null, wenn auch NK Null ist. Ist CA=1, so trennt ein Punkt die Nachkommastellen ab, bei CA=2 ist es das Komma und bei CA=3 ein Leerzeichen.

X$ schließlich ist die String-Variable, der der errechnete und konvertierte Ausdruck übergeben wird. Wurde X$ noch nicht zuvor im Basicprogramm verwendet, wird es neu angelegt, sonst wird der alte Inhalt gelöscht. Statt X$ können Sie eine beliebige Textvariable (etwa A$ oder TE$ usw.) einsetzen. Da der konvertierte Ausdruck ja formatiert ist, muß es sich logischerweise um eine Stringvariable handeln.

Das folgende Beispiel demonstriert die Anwendung: Es soll in einen Scheck der DM-Betrag eingesetzt werden. Dabei soll der Scheck verfälschungssicher sein, weshalb wir Sternchen vor die Summe setzen (FZ=42). Der Betrag soll 15 Zeichen lang sein (LE=15), das Vorzeichen steht hinter dem Betrag (VZ=128). Die Tausenderpunkte sind eingeschaltet (TS=1), als Trennzeichen für die Pfennige (NK=2) wählen wir das Komma (CA=2). Der Betrag wird in BE$ übergeben und dann mit PRINT ausgegeben:

```
10 INPUT"BETRAG IN DM";DM
20 SYS 51200,DM,15,128,1,2,42,2,BE$
30 PRINT"BETRAG: DM ";BE$
40 END
```

Vergessen Sie nicht, vor der Eingabe dieses Programms »USING« wie oben beschrieben zu laden! Einige Beispiele für Ein- und Ausgaben:

BETRAG	AUSGABE
12	DM *********12,00
23547.124	DM *****23.547,12
-77777777.777	DM *77.777.777,78-
0.50	DM **********0,50

Gültige Eurocheques, die mit dem C 64 und Using bedruckt wurden, nimmt der Autor jederzeit gern entgegen.

Damit dürfte Ihnen das Prinzip klar sein. Wollen Sie die Routine in eigene Programme einbauen, sehen Sie am besten wie folgte eine Boot-Zeile vor:

```
1 IF A=0 THEN A=1:LOAD "USING",8,8
2 weiter im eigenen Programm
```

Zum Schluß noch einige Hinweise. Bitte setzen Sie für die Parameter, vor allem für CA, keine anderen Werte als die oben angebotenen ein, Fehlfunktionen oder Fehlermeldungen wären die Folge. Die Fehlermeldung ?FORMULA TOO COMPLEX ERROR kann theoretisch ebenfalls erscheinen und weist dann auf eine interne Störung bei USING hin, das Programm ist jedoch sicher programmiert und weist keine Fehler mehr auf. Die Ausgabe des konvertieten String auf den Drucker ist mit OPEN4,4:PRINT#4,BE$ natürlich ebenfalls möglich, ebenso wie der von SYS 51200 erzeugte String ganz normal wie ein solcher behandelt und mithin auch z.B. mit LEFT$ usw. nachbearbeitet werden kann.

Warum nur muß man auf dem C 64 solche wichtigen und wertvollen Tools nur immer selbst programmieren, warum wird zum Beispiel keine eingebaute Using-Routine mitgeliefert?

4. KAPITEL: FLOPPY-PROGRAMMIERUNG

Relativ einfach!

Alles ist relativ, das hat schon der alte Albert Einstein gesagt. Aber viele Programmierer haben Angst vor relativen Dateien. Wir zeigen Ihnen, wie es einfach und schnell geht, welche Fallen beachtet werden sollten.

Die Idee, die hinter den relativen Dateien steckt, ist die folgende: Um beispielsweise den Eintrag (man sagt »Record«) Nummer 25 zu lesen, brauchen Sie sich nicht vorher durch die Records 1 bis 24 zu wühlen, so wie dies bei Verwendung der sequentiellen Dateien (SEQ) der Fall wäre. Ein weiterer Vorteil ist, daß wir keine Kopie der gesamten Datei machen müssen, um nur einen Datensatz zu ändern; es wird stattdessen einfach der alte Eintrag durch den neuen ersetzt - ganz unbürokratisch.

REL-Dateien im Prinzip...

Und wie funktioniert das intern? Zunächst einmal belegt jeder Eintrag einen bestimmten, beim Anlegen der Datei vorgegebenen Platz. Dadurch müssen die Einträge, die dem Datensatz folgen, den wir ändern wollen, nicht verschoben werden. Nicht der ganze Platz, der pro Datensatz vorgesehen wurde, muß jedoch auch belegt werden. Sie könnten eine Datei mit einer Datensatzlänge von 60 erzeugen, obwohl einige (oder alle) Einträge weniger als je 60 Bytes belegen. Die restlichen Bytes werden mit Nullen aufgefüllt. Der zweite Teil des Tricks betrifft die Einführung eines Zeigers, des »Index«. Immer, wenn Sie einen bestimmten Datensatz ansprechen, führt das Laufwerk verschiedene Berechnungen durch und weiß dann, an welcher Stelle auf der Diskette die gesuchten Daten stehen. Dazu existiert für jede REL-Datei ein sogenannter »Side-Sector« (zu deutsch: Zeigerblock), in dem die belegten Blocks verzeichnet sind. Die Zeigerblöcke fungieren als »Wegweiser« durch das File. Mit der genauen Artihmetik braucht sich der Programmierer zum Glück nicht befassen, sie ist ziemlich kompliziert. Rufen wir beispielsweise den Datensatz 15 einer relativen Datei mit Datensatzlänge 100 ab, weiß das Laufwerk aufgrund seiner Berechnungen, daß es dazu den 6. Sektor der Datei lesen muß. Es sucht im Side-Sector die Adresse (Track, Sektor) und liest den Block. Danach ergeben weitere Berechnungen, daß sich der gesuchte Datensatz ab Byte 130 in diesem Block befindet. Wieder braucht man sich keine Gedanken darüber machen, das haben bereits die Entwickler der 1541 getan (allerdings nicht immer ganz astrein, wie wir später sehen werden). Sie geben nur die Datensatznummer an und lassen den Computer rechnen. Weitere Hinweise zur internen Bearbeitung durch die Station finden Sie unten.

... und in der Praxis

Es gibt aber auch einige wesentliche Nachteile bei relativen Dateien. Sie sind langsamer, größer und komplexer als SEQ-Files, und halten eine ganze Reihe an bösartigen Fallen bereit, in die man unweigerlich stürzt, wenn man nicht einige Tricks kennt. Diese Ihnen vermitteln will dieser Artikel.

Betrachten wir ein SEQ-File. Es ist viel einfacher als eine REL-Datei, und erledigt vieles mindestens genauso gut. Eine alte Grundregel der Datenverarbeitung besagt: Falls pro »Session« auf mehr als 15 Prozent eines Files zugegriffen werden muß, sollte man eine sequentielle Datei verwenden; wenn weniger, eine relative Datei. Der Knackpunkt ist, daß es völlig unnötig ist, sich erst durch den Anfang der Datei zu kämpfen, wenn man einen weiter hinten liegenden Datensatz lesen will. In diesem Fall erlaubt die REL-Datei direkten Zugriff auf den Teil, den Sie suchen. Eine sequentielle Datei dagegen muß vom Anfang bis zum Ende gelesen werden, und eine Änderung verlangt die Erstellung einer neuen Kopie der Datei.

Ziehen Sie auch folgende Punkte in Betracht. Belegt eine sehr große sequentielle Datei sehr viel Platz auf Diskette (über eine halbe Diskettenseite), haben Sie gar nicht mehr genügend Platz für eine Kopie auf der selben Floppy. Hier muß man der relativen Datei einen klaren Vorteil einräumen. Einen wesentlichen Nachteil der relativen Datei allerdings sollten Sie immer im Auge behalten. Soll ein Datensatz geändert werden, geht der alte Inhalt unwiderruflich verloren. Ein Eingabefehler beispielsweise kann dazu führen, daß wichtige Informationen für immer verloren sind. Mit SEQ-Dateien erzeugen Sie die neue Version, indem Sie eine Kopie anlegen. Die alte Version ist, ggf. »unsichtbar«, immer noch vorhanden und könnte im Falle eines Falles wiederhergestellt werden.

Das Demoprogramm

Den Umgang mit REL-Dateien erlernen Sie am besten anhand eines fertigen Programmes. Das Programm zu diesem Artikel ist weitgehend selbsterklärend und nicht zu knapp mit REM-Vermerken dokumentiert, daher wollen wir im folgenden nur auf wichtige Eigenarten besonders hinweisen. Im Programm werden einige der folgenden Merkpunkte in die Praxis umgesetzt. Es müßte auf allen Acht-Bit Rechnern von Commodore mit einer der üblichen Diskettenstationen laufen. Im Textkasten finden Sie eine Zusammenstellung der wichtigsten Diskettenbefehle.

Die folgenden Richtlinien wollen als Sicherheitsratschläge verstanden werden. Manchmal macht es nichts aus, wenn man eine davon bricht. Jedoch lieben nur wenige von uns das Risiko, Daten zu verlieren, mithin kann es nicht schaden, die sieben Grundregeln sorgfältig, konsequent und stur zu beachten. Die meisten dieser Maßnahmen liegen übrigens in Betriebssystemfehlern im Laufwerks-DOS begründet, in das Commodore bekanntlich nicht allzu viel Mühe investiert hat. (Eine Sonntags-Produktion?)

Sieben Regeln

Regel 1: Erzeuge genügend Datensätze

Wenn Sie das erste Mal ein File anlegen, sorgen Sie dafür, daß ausreichend Records belegt werden, so daß mehr als ein Block auf Diskette von der Datei belegt wird. Das Beispielprogramm arbeitet etwa mit einer Datensatzlänge von 33, mithin sollten acht oder mehr Datensätze angelegt werden (254/33 = 8). In unserem Fall werden 10 Datensätze erzeugt (vgl. Zeile 320). Später kann

eine REL-Datei zwar theoretisch erweitert werden, jedoch klappt das in der Realität nicht immer reibungslos. Gönnen Sie der Datei daher lieber zu viele Einträge, wenn Sie sie anlegen.

Beim Anlegen der Datei in Zeile 290 (hier teilen wir dem Laufwerk das erste Mal mit, daß wir die Absicht haben, mit REL-Dateien zu operieren) stellen wir die Datensatzlänge auf 33 ein. Ihnen fällt vielleicht auf, daß diese Angabe bei den OPEN-Befehlen in 190, 390 und 1400 fehlt. Anders als bei SEQ-Dateien (hier müssen Sie immer schreiben: OPEN 2,8,2,"name,S,R") können wir uns diese Angabe beim Öffnen einer bereits bestehenden REL-Datei sparen.

Es kann nicht schaden, die Datei mit einem gesonderten Programm anzulegen. Nachdem ein File angelegt wurde, wird es im Folgenden nur noch »updated«. Im Beispielprogramm wird in den Zeilen 170 bis 240 geprüft, ob die Datei schon besteht. Wenn nicht, wird sie in 280 bis 390 neu angelegt. Das Programm akzeptiert als einzige Floppy-Fehlernummer die 62 (FILE NOT FOUND), alles andere ist ein »echter« Fehler, der vor dem Programmstop in den Diagnoseroutinen ab Zeile 1620 ausgewertet wird.

Regel 2: Setze den Record-Zeiger immer auf das erste Byte des Datensatzes

Der Zeiger im P-Befehl (vgl. Textkasten) sollte immer auf das erste Zeichen im Eintrag gestellt werden. Der letzte Parameter des P-Kommandos sollte daher CHR$(1) sein (siehe Testprogramm Zeilen 670, 920, 1300). Lesen oder beschreiben Sie jedes Mal den ganzen Record in einem »Aufwasch«, so ersparen Sie sich unnötigen Ärger.

Regel 3: Prüfe den Disk-Status

Lesen Sie immer den Status aus dem Fehlerkanal 15, nachdem Sie der relativen Datei einen Befehl gegeben haben. Sie brauchen den ermittelten Fehler überhaupt nicht weiter auszuwerten (Zeile 1360), durch das Lesen des Fehlerkanals gewinnt das Laufwerk die notwendige Zeit, um die gesuchte Stelle der Datei aufzuspüren und anzufahren.

Es ist interessant, daß einige Fehlermeldungen durchaus im Normalbetrieb auftreten und den korrekten Ablauf anzeigen. So erwartet Zeile 340 einen Fehler Nr. 50. Der Grund: Vorher wurde die relative Datei angelegt, indem wir das »Freizeichen« CHR$(255) in den höchsten Datensatz (Nr. 10) schrieben. Dieser Satz existierte noch nicht, da die Datei ja eben erst angelegt wurde. Das Laufwerk läßt dies natürlich nicht ungestraft mit sich machen, und reagiert etwas sauer, wenn auch zu unrecht mit dem Fehler. Bei der Statusprüfung in Zeile 690 weist ein NO RECORD-»Fehler« darauf hin, daß es in der Datei keine weiteren Einträge mehr zu lesen gibt. Das Programm bricht die Schleife hier ab. Und in Zeile 930 informiert eben dieser Fehler den Anwender darüber, daß der momentan beschriebene Datensatz noch »jungfräulich« war.

Fehlerkanal lesen

Sie denken bitte daran, daß das wichtige nur das Lesen des Status ist, es dient nicht nur zum Erkennen von Fehlern, sondern auch zur Verlangsamung des Programms, damit keine Daten zwischen Computer und Laufwerk verloren gehen, bevor das Drive fertig ist.

Regel 4: Addiere 96 zur Sekundäradresse

Es ist nicht weiter schwer, beim P-Befehl (z.B. in Zeile 320, 670, 920 oder

1320) zur Sekundäradresse den Wert 96 zu addieren, damit das Drive die Datei richtig behandelt. Diese wurde in Zeile 190 mit der normalen Sekundäradresse 2 geöffnet, zu der wir dann in den genannten Zeilen 96 addieren (ergibt 98), um auf die Datei zuzugreifen.

Regel 5: Nur ein PRINT# pro Datensatz

Wenn Sie in ein File schreiben, benutzen Sie einmal den Befehl PRINT#, um einen Eintrag zu schreiben; nicht mehr und nicht weniger. Sie sollten alle Felder des Satzes in einem »Aufwasch« schreiben. (Wenn Sie die weniger zuverlässige Methode verwenden, mit dem P-Befehl und einem letzten Parameter ungleich 1 auf ein beliebiges Byte innerhalb des Records zu positionieren, können Sie einen Teil des Eintrags verändern, sollten es aber aus Sicherheitsgründen nicht).

Sehen Sie sich die Zeilen 1000 bis 1080 an. Obwohl es in einem Datensatz verschiedene Felder gibt, werden sie in Zeile 1050 zusammengenagelt, damit wir sie in Zeile 1080 alle zusammen an die 1541 senden können. Die einzelnen Felder werden mit einem CR (Carriage Return, CHR$(13), entspricht der <RETURN>-Taste) getrennt, jedoch ist am Ende des Textes kein CR erwünscht. Daher wird dieses in Zeile 1070 ausgefiltert. Interessieren beim Lesen verschiedene Datenfelder, gehen Sie so vor wie in den Zeilen 620 bis 770, wo die System-Status-Variable dazu verwendet wird, festzustellen, ob innerhalb des Datensatzes noch weitere Felder folgen.

Wenn Sie, wie es in unserem Beispielprogramm der Fall ist, vorhaben, in sequentieller Reihenfolge auf die Einträge zuzugreifen (1, 2, 3, und so weiter), sollten die vorstehenden Maßnahmen genügen. Wollen Sie jedoch auch beim Lesen wild durch die Datei springen (etwa im Menüpunkt 2), folgen hier noch zwei Merkpunkte, damit es keine Bauchlandung wird:

Beliebige Reihenfolge

Regel 6: Zum Lesen zweimal positionieren

Wenn Ihr Programm Datensätze in beliebiger Reihenfolge liest, geben Sie das Positionier-Kommando (P) zweimal hintereinander. Lesen Sie jedes Mal den Status der Station. Das klingt nach »Dummsuff«, hat aber durchaus seinen guten Grund: Um auf einen neuen Datensatz zu positionieren, muß das Laufwerk oft einen neuen Datenblock einlesen. Dazu muß erst wieder der side sector gelesen werden, um die Track/Sektorangabe des nächsten Datenblocks zu finden. Manchmal, wenn der neue Datenblock sehr weit vom alten entfernt liegt, muß sogar ein neuer side sector gelesen werden. Schließlich müssen noch zwei eigentliche Datenblocks von der Magnetscheibe geholte werden, bevor Daten zur Verfügung stehen. Das alles geht natürlich nicht mit Hexerein, sondern kostet unter Umständen viel Zeit. Gönnen Sie dem Laufwerk eine Verschnaufpause und positionieren Sie zweimal. Das Demoprogramm tut genau dieses im Bereich 630 bis 700.

Regel 7: Zum Schreiben Datei erst einmal schließen und anschließend wieder öffnen

Schreibt das Programm Datensätze ohne Einhaltung der Reihenfolge, sollte das File zur Sicherheit nach jedem Schreibzugriff geschlossen und gleich wieder geöffnet werden (nach Auslesen des Status). Sie bezahlen dafür ggf. über eine Sekunden an Programmlaufzeit, umgehen aber den Fehler im Betriebssystem der Floppy, und stellen sicher, daß keine Daten im Nirwana landen.

Wenn Records geschrieben werden, hat das Laufwerk eine Menge zu tun. Es führt nicht jedes Mal tatsächlich auch Schreibzugriffe auf die Magnetschicht aus, wenn Sie den PRINT#-Befehl gehen, sondern speichert (theoretisch) die Daten in einem Puffer und wartet, bis ein CLOSE-Befehl erfolgt oder mit dem P-Kommando ein anderer Eintrag angewählt wird. Die Diskettenstation sollte dann eigentlich die geänderten Daten sicher speichern, bevor sie den nächsten Block liest. Dies ist allerdings eine etwas »kitzlige« Angelegenheit, manchmal kommt es vor, daß die 1541 schlicht und einfach vergißt, daß noch Änderungen ausstehen.

Die sicherste Vorgehensweise ist, das File zu schließen und wieder zu öffnen. Genau das tut unser Beispielprogramm in Zeilen 1110 mit 1160 und 1370 mit 1420. Sicher, in diesem Fall ist das wohl zu viel des Guten, da das Drive genügend Zeit zur Speicherung hat, während der Anwender den nächsten Datensatz eingibt. Aber es sind Ihre Daten - Sicherheit geht vor.

Noch mehr Fallen

Es gibt da noch einige andere Anormalitäten, wenn Sie mit REL-Files zu tun haben. Die Fehler im DOS sind gravierender, als man glauben möchte. Bitte wundern Sie sich bei Ihren Experimenten nicht, wenn im Fehlerkanal plötzlich nie dagewesene Fehlermeldungen wie

```
49,,00,00
08,DISK,00,00
04,OPEN,00,00
06,NOT,00,00
```

stehen (kein Witz - diese und andere Meldungen traten bei der Recherche zu diesem Text tatsächlich auf!).

Auf Commodore-Floppies sollten relative Dateien nicht die ASCII-Zeichen 0 (Nullbyte) und 255 enthalten. Versuchen Sie, eines dieser Zeichen in einen Record einzubauen, werden Sie wahrscheinlich unangenehme Überraschungen erleben. Zeichen 0 wird verwendet, um den nicht belegten Teil des Eintrags aufzufüllen. Steht ein Nullbyte zwischen den Daten, kann die 1541 nicht mehr feststellen, wo der Datensatz endet, und arbeitet falsch. Die 255 wird vom DOS verwendet, um einen leeren Datensatz zu markieren - ohne Daten. Unser Beispielprogramm schreibt diesen Code in Zeile 350 beim Anlegen der Datei, in Zeile 1350, um einen Eintrag zu löschen und wertet ihn in 720 aus, um einen leeren Datensatz zu erkennen. Wollen Sie diesen Code verwenden, passen Sie also auf.

Relative Files mit Datensatzlängen 42, 58 oder 63 können von der 1541 nicht angelegt werden. Damit müssen Sie sich einfach abfinden. Grund: Aufgrund eines Fehlers im DOS wertet das System diese drei Zahlen als die ASCII-Codes für Stern, Doppelpunkt und Fragezeichen, und solche Zeichen sind für Disk-Befehle bekanntlich tabu.

Sie sollten REL-Dateien grundsätzlich nur auf leeren Disketten anlegen, da sie (wenn sie groß genug werden) die anderen Dateien auf der Diskette und schlimmstensfalls sogar das Directory ohne Rücksicht auf Verluste gnadenlos überschreiben. Auch das Erweitern einer solchen Datei (laut Handbuch »problemlos möglich«) ist daher nur mit Vorsicht zu genießen.

Sie verfügen jetzt über das notwendige Werkzeug, um mit relativen Dateien arbeiten zu können. Was vielleicht noch fehlt, ist das Verständnis für die internen Abläufe, die uns als Programmierer im Normalfall gar nicht zu

interessieren brauchen. Dennoch kann es nicht schaden, sich kurz damit zu befassen - vor allem auch, wenn es darum geht, den Platzbedarf eines solchen Files abzuschätzen.

REL-Dateien intern

Wie bereits erwähnt, besteht die REL-Datei aus drei Teilen. Da wäre zum einen der Eintrag im Directory. Zum zweiten die normalen Datenblöcke, die ganz normal wie bei einer SEQ-Datei verkettet sind. Die Side-Sectors (Zeigerblöcke) bilden den dritten Teil. Sie dienen als Wegweiser durch die Datei. Jeder Side-Sector enthält in den ersten beiden Bytes einen Zeiger auf den nächsten Side-Sector. Byte 2 enthält die Nummer des Side-Sectors (0 bis 5), in Byte 3 steht die Datensatzlänge. Danach folgen in zwölf Bytes die Adressen (Track/Sektor) aller maximal sechs Side-Sector-Blöcke einer REL-Datei (siehe unten). In den verbleibenden 256 - 2 - 2 - 12 = 240 Bytes stehen die Adressen von maximal 120 Datenblocks. Dadurch, daß in jedem Side-Sector die Adressen aller anderen Side-Sectors stehen, muß das Laufwerk nur einen Lesezugriff ausführen, um einen beliebigen Zeiger zu lesen - das bringt einen Geschwindigkeitsvorteil. Pro 120 angefangenen Datenblöcken wird also ein Side-Sector angelegt. Für eine neu formatierte Diskette, die 664 freie Blöcke zur Verfügung stellt, ergibt sich demnach ein Bedarf von maximal sechs Zeigerblöcken, dadurch bleiben bis zu 658 Blöcken zur relativen Datenspeicherung übrig. Die kleinstmögliche REL-Datei besteht aus zwei Blöcken (ein Side-Sector, ein Datenblock). Jeder Datenblock enthält 254 Datenbytes (die ersten beiden Bytes werden für die Verkettung benötigt, mithin stehen auf einer Diskette maximal 167132 Bytes für die relative Datenspeicherung zur Verfügung. Wir errechnen die maximal mögliche Anzahl von Einträgen bei gegebener Datensatzlänge, indem wir die 167132 durch die Satzlänge teilen. Auch hier gilt die theoretische Obergrenze von 65535 Sätzen (dies entspricht der höchsten mit zwei Bytes darstellbaren Zahl), die überhaupt nur bei einer Datensatzlänge von zwei Zeichen übschritten werden könnte. Unsere Beispieldatei erlaubt bis zu 167132 / 33 = 5064 Einträge. Bei der höchstzulässigen Datensatzlänge von 254 Bytes erhalten wir genau 658 Datensätze, entsprechend einem Satz je Block.

Bei der Arbeit mit REL-Files muß der Programmierer also einige Dinge besonders beachten, die auf den ersten Blick nicht logisch erscheinen. Aber wenn Sie die Regeln beim Programmieren einhalten, werden Sie keinerlei Probleme haben. Sie werden mit einer sehr flexiblen Art der Datenspeicherung belohnt, die nicht nur Programme professioneller macht, sondern auch große Vorteile in Komfort und Geschwindigkeit mit sich bringt.

Literatur:
Immer der Reihe nach, 64'er 7/89, Seite 94
Kreuz und quer, 64'er 8/89, Seite 56
Kreuz und quer, 64'er 9/89, Seite 98
Floppy-Flops, 64'er 1/91, Seite 55
Floppy-Flops, 64'er 2/91, Seite 50
Karsten Schramm, Die Floppy 1541, Markt & Technik Verlag, Best.Nr. 90444
Lothar Englisch, Gerd Szczepanowski, Das große Floppy-Buch, Data Becker

Befehle zu relativen Dateien:

(1) Berechnung von Low- und Highbyte
```
- Highbyte berechnen       HB=INT(SN/256)
- Lowbyte berechnen        LB=SN-HB*256
```

(2) Anlegen einer REL-Datei

```
- Befehlskanal öffnen      OPEN 1,8,15
- Datenkanal öffnen        OPEN 2,8,2,"name,L,"+CHR$(SL)
- Positionierung auf höchsten Datensatz
                           PRINT#1,"P"CHR$(98)CHR$(L)CHR$(H)CHR$(1)
- Freigeben des Satzes     PRINT#2,CHR$(255);
- Kanäle schließen         CLOSE 2:CLOSE 1

(3) Öffnen einer bestehenden REL-Datei
- Befehlskanal öffnen      OPEN 1,8,15
- Datenkanal öffnen        OPEN 2,8,2,"name"

(4) Lesen eines Datensatzes
- Positionierung auf den Datensatz
                                PRINT#1,"P"CHR$(98)CHR$(L)CHR$(H)CHR$(1)
- Daten einlesen           INPUT#2,daten oder GET#2, daten

(5) Schreiben eines Datensatzes (alle Daten in einem String!)
- Positionierung auf den Datensatz
                                PRINT#1,"P"CHR$(98)CHR$(L)CHR$(H)CHR$(1)
- Daten schreiben          PRINT#2,daten
- Kanäle schließen         CLOSE 2:CLOSE 1
```

Abkürzungen: SN = Satznummer, H = Highbyte, L = Lowbyte, SL = Satzlänge

Der P-Befehl

```
PRINT#1,"P"CHR$(c)CHR$(l)CHR$(h)CHR$(p)
```

CHR$(c) übergibt die Sekundäradresse des OPEN-Befehles, unter dem die REL-Datei geöffnet wurde, plus 96
CHR$(l) und CHR$(h) übergibt im Format Lowbyte, Highbyte die Nummer des Datensatzes, auf den positioniert werden soll
CHR$(p) kennzeichnet die Stelle, auf die innerhalb des Eintrags positioniert werden soll. p sollte 1 sein.

ARC 1.5 - ein ganz neues Tool für den C 64

Aus der Welt der Personal Computer ist es nicht mehr wegzudenken. Nun wird auch der C 64 davon befallen. ARC schlägt zu!

Vielleicht wissen Sie, was ein »Linker« ist: Dieses Programm gestattet es, ein mehrteiliges Programm, das seine Bestandteile nacheinander in den Speicher lädt (»nachlädt«), zu einem File zusammenzufassen, das dann ohne Nachladen auskommt.

Das einteilige Programm, das dabei erzeugt wird, startet man mit RUN. Es werden jetzt die einzelnen Bestandteile, die der Linker direkt aneinandergehängt hat, wieder an ihre richtigen Positionen im Speicher geschoben und dann das Programm normal gestartet. Natürlich muß man bei dieser Methode die Nachladebefehle aus dem Hautprogramm entfernen, da dieses sonst nun versucht, ganz normal nachzuladen.

Bislang gab es jedoch noch kein Programm für den C 64, das zwar auch

verschiedene Files von Diskette zusammenfügt und als Gesamtprogramm speichert, das nach dem Starten dann aber die einzelnen Files wieder auf Diskette generiert. Vor allem auf PCs, aber auch beim Amiga sind solche Tools, »Archive« genannt, Standard. Man verwendet sie etwa, wenn man sehr viele kleine Programme beispielsweise über BTX oder das Telefon (DFÜ) übertragen will. Hier ist es leichter (und billiger), nur ein kompaktes Gesamtpaket zu übertragen. Der Empfänger startet dieses dann, legt eine leere formatierte Diskette ein und die kleinen Files werden auf Diskette generiert.

Auch zur Datensicherung sind solche Utilities geeignet. Stellen Sie sich vor, Sie haben so um die vierzig kleine Files zu je etwa vier Blocks, die Sie beispielsweise auf Cassette überspielen wollen. Danach wandert die Cassette ins Archiv, um die Programme sicher aufbewahrt zu haben.

Sie könnten nun alle vierzig Programme nacheinander auf das Band überspielen. Jedoch würde vielleicht ein Packvorgang enormen Platzgewinn bringen. Das Packen jedes einzelnen Files ist jedoch nicht nur sehr arbeitsaufwendig, sondern auch nicht effizient, da hier der Entpacker schon so lang würde, daß der durch Packen erzielte Platzgewinn zumindest wieder ausgeglichen werden würde. Packen Sie nun mit einem »Archive«-Utility alle vierzig Files zu einem Programm zusammen, es würde dann so um die 162 Blocks lang werden (etwa zwei Blocks nimmt die kleine Routine in Anspruch, die nach dem Starten des Kompaktfiles die einzelnen Files wieder auf Diskette generiert). Durch Packen läßt sich dieses sicherlich auf weit unter 162 Blocks quetschen, und Sie erreichen enormen Platzgewinn.

Aber so ein Utility gab es für die kleinen Commodore-Computer bisher noch nicht. Daher wurde »ARC« geschrieben. Dieser Spezial-Linker unterscheidet sich, wie gesagt, von herkömmlichen Linkern dadurch, daß nach dem Start des Gesamtfiles nicht (nur) im Speicher, sondern auf Disk die einzelnen Files wieder erzeugt werden. Das in reiner Maschinensprache geschriebene Tool kann dabei maximal 45 Stück verarbeiten. Sollen es noch mehr sein, müssen Sie die ersten 45 Files zusammenpacken, den Rest in ein zweites Gesamtfile, die beiden Kompaktfiles durch einen guten Packer schicken (»ARC« packt nicht automatisch, da Packer laufend weiterentwickelt werden, der Anwender kann dann einen Packer seiner Wahl anwenden), und die dabei entstehenden Files noch einmal zusammen-manschen.

Das Programm wird mit dem Befehl

LOAD "ARC 1.5",8

geladen und mit

RUN

gestartet. Nach dem Start sind zunächst die Namen der Einzelfiles einzugeben. Die Eingabe ist jedes Mal mit <RETURN> zu beenden. Sämtliche Eingaberoutinen des Programmes sind so programmiert, daß durch Cursorbewegungen die Bildschirmmaske nicht zerstört werden kann. Bitte vermeiden Sie bei der Eingabe Joker (»*« und »?«), um die Filenamen abzukürzen. Beim Laden gibt es zwar keine Probleme, da beim späteren Speichern jedoch wieder exakt die selben Namen verwendet werden, ist die Verwendung von Jokern nicht erlaubt.

Durch Eingabe des Dollarzeichens an erster Stelle wird das Inhaltsverzeichnis der eingelegten Diskette (Directory) auf dem Bildschirm

gelistet. Am Ende kann durch Tastendruck wieder in die Eingabeschleife zurückgekehrt werden. Hinter dem Dollarzeichen kann die Directory auch genauer spezifiziert werden. So listet etwa »$HA*« alle Files, die mit »HA« beginnen.

Durch Eingabe eines Fragezeichens können Sie die letzte Eintragung korrigieren. Dies funktioniert natürlich nicht, wenn Sie erst bei der Eingabe des ersten Namens sind, oder der Cursor in der obersten Zeile steht. Die Liste der Filenamen wird bei der Eingabe gescrollt.

Zur Beendigung geben Sie einfach einen Stern ein. Dies funktioniert nur, falls bereits mindestens ein Filename eingegeben wurde.

An Stelle des 46. Filenamens nimmt das Programm nur noch das Dollarzeichen, das Fragezeichen und den Stern an, da mehr Filenamen nicht erlaubt sind.

Für den Fall, daß Sie sehr viele Routinen mit ähnlichen oder schwer zu merkenden Namen zusammenpacken wollen, ist es sinnvoll, die gewünschten Filenamen direkt aus dem Directory zu übernehmen. Daher wurde in der neuen Version 1.5 eine solche Funktion implementiert. Geben Sie dazu anstelle eines Filenamens einen Doppelpunkt ein und drücken <RETURN>. Die Bildschirmmaske wechselt nun, Sie werden aufgefordert, eine Diskette einzulegen. Legen Sie die Floppy mit den Quellprogrammen ein und drücken eine Taste. Mit <A> wird ARC 1.5 völlig neu gestartet (Datenverlust!). Jetzt zeigt das Tool den Diskettennamen an. Danach erscheinen der Reihe nach alle gespeicherten Files mit der dazugehörigen Länge in Blocks (254 Byte). Mit den Tasten <J> und <N> (für »ja« und »nein«) wählt man, ob das angezeigte File in das Gesamtfile übernommen werden soll. Auch in diesem Modus sind nur maximal 45 Files möglich, mehr faßt der Namens-Speicher nicht. Das Programm zeigt laufend die Anzahl der bereits gewählten Files an. Wenn Sie mindestens ein File selektiert haben, beenden Sie die Auswahl, wenn Sie fertig sind, mit <*>. Sollen nur die ersten Files eines Directories erfaßt werden und danach eine neue Diskette eingelegt werden, können Sie nach Druck auf <D> eine neue Diskette einlegen. Die Taste <A> dient zum Abbruch, hier wird ARC 1.5 neu gestartet, alle bisher erfaßten Namen werden vergessen.

Während der automatischen Namenerfassung aus dem Directory zeigt ARC 1.5 ständig die Anzahl der geschätzten von den selektierten Dateien belegten Blocks und die Anzahl der freien Blocks an. Dadurch wird weitgehend vermieden, daß Sie später beim Einlesen eine böse Überraschung erleben: »Speicher voll«. Der Speicher für die Files erstreckt sich von $9cd bis $bfff, das sind ca. 181 Blocks. Wird diese Anzahl überschritten, können mit <J> keine weiteren Files mehr selektiert werden. Die Länge der einzelnen Files wird übrigens dem Directory entnommen. Da diese grobe Abschätzung allerdings vor allem bei sehr kurzen Dateien (nur wenige Bytes) ungenau wird (Rundungsfehler), ist es möglich, auch bei angezeigtem Speicherüberlauf weitere Files mit <SHIFT J> zu übernehmen, auf die Gefahr hin, daß später beim Einlesen »Speicher voll« erscheint. Beispiel: Der Zähler zeigt 166 belegte und 15 freie Blocks (Summe 181). Jetzt können mit <J> nur noch Files mit 15, 14, 13 usw. Blocks Länge aufgenommen werden, keine Files über 15 Blocks Länge. Diese lassen sich nur mit <SHIFT J> erfassen. Im Normalfall sollten Sie die Abschätzung allerdings als obere Grenze akzeptieren, mit einer weiteren Einschränkung: In diese Berechnung gehen keine Files ein, die im ersten Modus (manuelle Eingabe der Filenamen) erfaßt wurden. Haben Sie den Automatik-Modus erst aktiviert, nachdem Sie schon einige Files von Hand eingegeben haben, werden diese nicht mitgerechnet, der Speicher kann dann u.U. sogar weniger als 181 Blocks fassen. Es ist nicht, auch nicht mit <SHIFT J>, möglich, die Obergrenzen von 45 Files zu überschreiten.

Vom Modus des automatischen Einlesens können Sie nicht in den Modus der manuellen Eingabe zurückzukehren, wohl aber umgekehrt. Soll das Gesamtfile also Files enthalten, deren Namen Sie manuell über Tastatur eingeben wollen (Modus 1) und weitere, deren Namen Sie einlesen lassen wollen (Modus 2), so sind zuerst in Modus 1 die »manuellen Dateinamen« einzugeben, danach durch Eingabe des Doppelpunktes Modus 2 zu aktivieren und die restlichen Namen zu spezifizieren. Danach kann mit der Stern-Taste der Vorgang beendet werden.

Bitte beachten Sie noch, daß es in Modus 2 unter Umständen Probleme bei Directory-Manipulationen geben kann, etwa dann, wenn nach dem zweiten Anführungszeichen noch Ladehilfen wie »,8,1« stehen, oder wenn der Filename im Directory Cursor-Steuerzeichen oder Joker (»*« oder »?«) enthält. Insbesondere könnten Schwierigkeiten auftreten, wenn die im Directory verzeichnete Blocks-Länge nicht mit der tatsächlichen File-Länge übereinstimmt, da dann die oben erklärte Abschätzung falsch arbeitet, oder wenn Files auf der Diskette vorhanden sind, deren Länge mehr als 255 Blocks beträgt (kommt im Normalfall nicht vor, solch lange Files lassen sich mit ARC gar nicht bearbeiten). Wenden Sie Modus 2 nur bei Disketten an, die nicht zum Beispiel mit dem »Disc-Wizzard« manipuliert wurden!

Nach der Eingabe aller Namen ist zu wählen, ob alle Programm auf der selben Diskette stehen, oder ob nach jedem File auf einen Tastendruck gewartet werden soll, um Zeit für den Diskettenwechsel zu haben. Antworten Sie hier einfach mit <J> oder <N>.

Die folgenden Frage erwartet den Filenamen, unter dem das gelinkte Gesamtfile abgespeichert werden soll. Hier führt die Eingabe eines Sternes oder eines Fragezeichens zurück zur Frage, ob alle Files auf einer Diskette gespeichert sind. Bitte vermeiden Sie auch hier die Eingabe von verbotenen Zeichen, wie »*«, »?«, »=«, »:«, »$« und so weiter.

Danach liest »ARC« alle benötigten Files in den Speicher. Hierbei können Diskettenfehler auftreten (z.B. »FILE NOT FOUND«), die angezeigt werden. Der Anwender kann dann durch Druck auf die Taste <V> erneut versuchen, das File (eventuell von einer anderen Diskette) zu laden, mit <N> dieses File »vergessen« und mit dem nächsten fortfahren oder mit <E> den Abbruch des Einlesevorganges veranlassen. Dann wird sofort das Gesamtfile gespeichert.

Den eingelesenen Programmen steht insgesamt ein Speicherplatz von etwa 45 KByte zur Verfügung, dennoch kann es vorkommen, daß ein File zu lang ist. Auch dieser Fall wird abgefangen. Hier kann gewählt werden, ob das File zu überspringen oder der Einlesevorgang abzubrechen ist.

Der Bereich, der die Programme aufnimmt, reicht von $09CD bis $BFFF. Pro File ist als Länge zu rechnen: Länge des Files in Zeichen + 5 Verwattungsbytes (Nullbyte, je zwei Bytes Länge und Startadresse) + Länge des Filenamens in Zeichen. In $0801 bis $09CB findet sich das Maschinenprogramm zum »Entwirren«, in $09CC steht die Anzahl der enthaltenen Files.

Nach dem Einlesen wird das Gesamtfile auf Diskette gespeichert. Hier werden Sie auf alle Fälle aufgefordert, eine (neue?) Floppy einzulegen, diese Aufforderung muß wie üblich mit einem Tastendruck bestätigt werden. Tritt ein Fehler beim Speichern auf, kann gewählt werden, ob »ARC« aufgeben soll, oder ob die Speicherung zu wiederholen ist. Am Ende erscheint dann noch die Frage, ob Sie »ARC« noch einmal starten wollen (Taste <J>), oder ob ein Reset ausgelöst werden soll (<N>).

Das erzeugte Gesamtfile, das sich natürlich auch problemlos etwa auf Cassette speichern läßt, können Sie ganz normal wie ein Basicprogramm laden und nach Einlegen einer Diskette mit genügend freiem Platz mit RUN starten. Jetzt werden automatisch die Einzelfiles mit ihrem Originalnamen wieder erzeugt. Das Programm zeigt dabei die Nummer des gerade generierten Files an.

Es bleibt nur noch zu sagen, daß sowohl das Generatorprogramm »ARC« wie auch die Routine im Gesamtfile als Gerät die zuletzt aktive Diskettenstation ansprechen. Dazu wird die aktuelle Geräteadresse geprüft. Liegt sie nicht zwischen 8 und 15, so wird sie automatisch auf 8 gesetzt.

Das war eigentlich alles, was es zu »ARC 1.5«, einem Utility ganz neuer Machart, zu sagen gab. Der geneigte Leser findet in der Tabelle die Speicherbelegung des Tools, hilfreich etwa dann, wenn Ergänzungen oder Erweiterungen vorgenommen werden sollen. Wir wünschen Ihnen viel Erfolg beim Zusammenfügen Ihrer Programme!

Speicherbelegung hexadezimal (Version 1.5)

0002-0003	Zeiger in Speicher
0004	Anzahl der zu erzeugenden Programme
0005	Anzahl der Files (Generator)
008b	Flag: Files auf einer Diskette
008c	lfd. Nummer
008d	wirkliche Anzahl der erzeugten Files
008e-008f	letzter Stand von 0002-0003
00a6-00a7	temporär
00aa-00ab	aktuelle Länge (Byte)
00b2-00b3	Zeiger auf Länge
00b5-00b6	temporär
00fb-00fc	File-Länge
00fd	Summe der File-Längen (Blocks, max. 181)
00fe	momentane File-Länge (Blocks)
02c0-02ff	Disketten-Fehlermeldung
0312-0313	Anzahl Bytes des Files
0334-03ff	Filename (Eingabe)
0400-07ff	Bildschirm
0801-bfff	frei für Generator-Datei
0801-09cb	Generator
09cc	Anzahl der Files
09cd-bfff	Files-Datenbereich
c000-ccbe	ARC 1.5
c000-c002	Sprung auf Start c793
c003-c416	Texte
c417-c5e1	Kopf (Original-Generator)
c417-c4fc	Maschinenprogramm
c4fd-c5e1	Texte
c5e2-ccbe	Programm ARC 1.5: 100% Assembler
c5e2	Kopf ab 0801 erzeugen
c5ff	Dateinamen eingeben
c66c	Dateinamen-Speicher suchen
c6bd	Directory anzeigen
c793	Start Hauptprogramm
c7e5	Namen manuell eingeben
c855	fertig
c8ba	Files einlesen

```
c936      Fehlerkanal lesen
c9dc      Ladeschleife
ca1c      Abbruch
ca2f      alle Files eingelesen
caed      Files aus Directory einlesen
cb2b      Disk-Wechsel
cc7f      Block-Anzahl ausgeben
ccbe      letztes Programmbyte
cd00-cfff Speicher für max. 45 Filenamen, je max. 17 Zeichen
d000-d3ff Bildschirm-Zwischenspeicher für Directory
```

Anleitung zum Programm »Bumpmaster«

Was ist ein »Bump«? Die Floppy 1541 hat die Angewohnheit, wenn es ihr nicht gelingt, von einer Diskette etwas zu lesen, bevor sie eine Fehlermeldung ausgibt, es erst einmal mit »Rattern« zu versuchen. Bei diesem sog. »Bump«, den Sie sicherlich auch schon einmal erleben durften, wird der Schreib/Lesekopf im Laufwerk an eine ganz bestimmte Position gebracht, in der Hoffnung, daß ein Lesen von hier aus möglich ist. Der Nachteil an diesem Rattern ist aber, daß es nur selten zum Erfolg führt und eher den Vorgang unnötig verzögert. Schlimmer noch, das Rattern ist schädlich für die Laufwerksmechanik. Tritt der Bump zu oft auf, kann der Tonkopf dejustiert werden. Zur Probe können Sie das Rattern auch manuell erzeugen: Entfernen Sie die Diskette aus dem Drive, schalten Sie selbiges an und wieder aus und schicken Sie dann den INIT-Befehl: OPEN 1,8,15,"I":CLOSE 1.

Es bietet sich daher an, den Bump einfach abzuschalten, zu »verbieten«. Die Floppy ist im Prinzip ein eigener Computer mit eigenem Prozessor, eigenen Speicherzellen, eigener Zeropage. In Speicherzelle 106 des Floppy-RAMs (!) gibt das 7. Bit an, ob der Bump erlaubt oder verboten ist. Wird es gesetzt, gibt die Floppy ggf. sofort die Fehlermeldung aus, ohne zu rattern. Da das Bit nur mit verhältnismäßig großem Aufwand verändert werden kann, gibt es das Programm »Bumpmaster«. Laden und starten Sie es wie ein Basicprogramm. Es prüft nun, ob in der Floppy bereits der Schutz aktiviert ist. Danach können Sie mit der Leertaste das Rattern »ein- und ausblenden«. Durch <RETURN> wird der neue Zustand übernommen, er hält so lange vor, bis Sie das Drive abschalten, einen Reset auslösen oder - natürlich - mit dem Bumpmaster eine neue Veränderung vornehmen.

Anleitung zu »DON'T REPLACE!«

Das nur einen Block kurze Utility repariert den Replace-Programmierfehler des Floppy-Laufwerkes 1541 für die Befehle SAVE und OPEN.

Sie wollen eine neue Version Ihres Programmes auf Diskette speichern. Dazu wählen Sie den Dateinamen der Vorgängerversion. Das Laufwerk rebelliert: FILE EXISTS! Klar, es dürfen sich niemals zwei Programme unter dem gleichen Namen auf einer Floppy befinden. Also probieren Sie es mit dem Klammeraffen, dem Replace-Befehl:

```
SAVE "@:NAME",8
```

wobei das Zeichen @ für den Klammeraffen steht. Die 1541 soll erst das alte File löschen und dann das neue unter dem Namen »NAME« speichern. Leider funktioniert das nur in der Theorie, denn bekanntlich steckt im Betriebssystem des Laufwerks ein Programmierfehler, der sich bei Verwendung des Replace-Befehles (Klammeraffe) auswirkt: Manchmal kommt es vor, daß das falsche Programm gelöscht und das neue falsch im Directory eingetragen wird. Daher, so lernt jeder Anfänger, sollte der Klammeraffe-Befehl nicht benutzt, sondern erst mit Scratch das alte File und dann die neue Datei gespeichert werden.

Das Utility »DON'T REPLACE!« erledigt dies für Sie automatisch, im Direktmodus oder programmgesteuert. Sie verwenden wieder den Klammeraffen, der aber von diesem Utility abgefangen und automatisch in einen Scratch-Befehl umgewandelt wird. Im Programm-Modus merken Sie nichts davon, wohl aber im Direktmodus, denn es erscheint eine neue Meldung:

```
SAVE "@:FILENAME",8

SCRATCHING FILENAME
SAVING FILENAME
```

Der Rechner zeigt also an, daß jetzt erst die alte Version gelöscht wird. Der Klammeraffe gelangt gar nicht bis zum Laufwerk und kann so keinen Schaden anrichten. Eine echts Neuheit an »DON'T REPLACE« ist, daß es nicht nur beim SAVE-Befehl (sei es in Basic oder in Assembler) wirkt, sondern auch dann, wenn Sie beim OPEN-Befehl (oder beim Aufruf der OPEN-Routine in Maschinensprache) den Klammeraffen setzen. Drei Dinge sollten Sie beachten: Die Umwandlung erfolgt nur, wenn das angesprochene Gerät tatsächlich eine Floppy ist (Geräteadresse größer als sieben), wenn hinter dem Klammeraffen ein Doppelpunkt folgt (die Anzahl der Zeichen zwischen Klammeraffe und Doppelpunkt ist beliebig, es sind also auch Konstruktionen wie

```
SAVE "@0:NAME",8
```

erlaubt, für Anwender von Doppel-Laufwerken). Außerdem dürfen Sie den Klammeraffen wie gezeigt nicht verwenden, wenn der OPEN-Befehl ein File nicht zum Schreiben, sondern zum Lesen öffnet. Denn sonst würde erst das File gelöscht und anschließend normal zum Lesen geöffnet werden, was einen FILE NOT FOUND ERROR zur Folge hätte.

Die Anwendung von »DON'T REPLACE« ist denkbar einfach. Laden Sie einfach vor einer Computer-Session das Programm in den Speicher:

```
LOAD "DON'T REPLACE!",8,1
NEW
```

und starten es mit

```
SYS 49152
```

Mit diesem Befehl kann es auch nach einem Reset ggf. wieder aktiviert werden. Eine Einschaltmeldung erscheint, und die Erweiterung ist fortan aktiv. Sie kann auch mit

SYS 65418

ohne weiteres jederzeit wieder abgeschaltet werden. Bemerkung: Diese Erweiterung eignet sich nicht für die Datasette, da diese weder über den fehlerhaften Replace-Befehl, noch über Scratch verfügt. Besitzer anderer Laufwerke als der 1541 sollten es auf einen Versuch ankommen lassen.

Anleitung zum »Twin Search System, Version 2.1«

Dieses nur 15 Blocks kurze Utility gestattet auf sehr einfache und komfortable Weise die Suche nach Programmen, die (ggf. auch unter verschiedenen Namen) auf einer oder mehreren Disketten mehrfach vorkommen. Viele Sonderfunktionen machen dieses Programm für alle C 64 Anwender interessant.

Die Anwendungsgebiete für das Programm sind vielfältig. Beispielsweise sind die sogenannten »Bibliotheken« weit verbreitet. Auf einer oder mehreren Disketten werden zum Beispiel Sprites gesammelt. Im Laufe der Zeit sammeln sich da dann mehrere hundert Sprites an, und bald verliert man den Überblick, ob man nicht vielleicht einige Sprites doppelt, weil zum Beispiel aus verschiedenen Quellen stammend, archiviert hat. Das selbe gilt zum Beispiel für eine Diskette, auf der Zeichensätze gesammelt werden, die aus kommerziellen Programmen ausgebaut wurden. Erwischt man zwei Programme, die den selben Zeichensatz verwenden (soll gar nicht so selten vorkommen!), und kann sich nicht mehr daran erinnern, daß man diesen Font ja bereits vor einiger Zeit archiviert hat, so verschwendet man neun Blocks: Der Satz ist doppelt gespeichert. Auch könnten Sie mit dem »Twin Finder« (Zwillingsfinder) ja einfach einmal alle Ihre Utility-Disketten durchgehen. Vielleicht entdecken Sie ja, daß Sie bestimmte Programme auf mehreren Disketten gespeichert haben. Insgesamt eignet sich das Programm also sehr gut dazu, Diskettenspeicherplatz zu sparen, da die Duplikate gelöscht werden können. Sie können das »TSS« aber auch »nur« dazu einsetzen, alle Files anzeigen zu lassen, die gleich lang sind und/oder (lediglich) die selbe Startadresse haben.

Grundsätzlich könnte man dazu wie folgt vorgehen: Man lädt ein Programm nach dem anderen in den Speicher und vergleicht dieses nacheinander mit allen Programmen auf der Diskette Byte für Byte. Aber das kostet viel, viel Zeit, da jedes Programm nicht nur einmal, sondern zum Beispiel bei 20 Files 20 mal vollständig gelesen werden muß! Die Zeit, die diese Methode beansprucht, steigt quadratisch mit der Anzahl der zu testenden Files. Etwas schneller geht es, wenn Sie alle Files auf der Diskette ein einziges Mal in den Speicher des C 64 einlesen und dann im RAM miteinander vergleichen. Nur, wie sollen Sie zum Beispiel 150 HiRes-Grafiken zu je 32 Blocks (= gesamt 4800 Blocks oder 1200 Kilobyte) in den 64 K Speicher des C 64 einlesen?

Das »Twin Search System« arbeitet hier mit einem ebenso einfachen wie »genialen« Trick: Es werden zwar alle Files »gescannt«, also Byte für Byte gelesen. Aus den Programmbytes bildet das Tool dann jedoch auf vier verschiedene Weisen vier Prüfsummen, nur diese werden zusammen mit Filenamen, Startadresse und Länge des Files im RAM abgelegt. Nachdem alle Files gescannt sind, vergleicht das Programm dann nur diese Prüfsummen. Haben zwei Programme die selben vier Prüfsummen und sind sie zudem gleich lang, so kann man davon ausgehen, daß es sich um zwei identische Files handelt. Durch geschickt gewählte Algorithmen (siehe unten) wird verhindert, daß zum Beispiel die Zahlenkombinationen 24, 35 und 35, 24, die ja summiert den selben Wert 59 ergeben, als identisch angesehen werden. Die Wahrscheinlichkeit, daß zwei verschiedene Files trotzdem die selben vier Prüfsummen haben und zudem noch gleich lang sind, geht gegen Null. Dies kann mathematisch aufgrund der Berechnungsverfahren sogar bewiesen werden. Dieser Fall trat bei einer Großzahl von getesteten Programmen nicht ein einziges Mal auf.

Laden Sie das in reiner Maschinensprache geschriebene »TSS« mit dem Befehl

LOAD "TSS 2.1",8

und starten Sie es wie ein Basicprogramm mit RUN. Der Befehl LIST bringt hier keine Befehle zum Vorschein.

Nach einem Programmabbruch, dem Programmende oder einem Ausstieg mit <RUN STOP> <RESTORE> oder Reset kann das »TSS« mit SYS 2257 wieder aktiviert werden, so es sich noch im Speicher befindet.

Auf dem Bildschirm ist jetzt das Titelbild zu sehen. Legen Sie die erste Diskette mit zu testenden Files ein und drücken eine Taste. Jetzt soll die Directory-Maske eingegeben werden. Dazu gehen Sie so vor wie beim Scratch-Befehl: Die Joker »?« und »*« dürfen gesetzt werden. Die Eingabe von »OP.*« übernimmt alle Files, deren Namen mit »OP.« anfangen. »M??ER« übernimmt zum Beispiel »MAYER«, »MEIER«, »MEYER«, »MJWER« und so weiter. Auch der Filetyp kann angegeben werden, hinter dem Namen und einem Gleichzeichen: zum Beispiel selektiert »H*=P« alle PRG-Files, deren Namen mit H beginnen. Die Trennung verschiedener Masken mittels Kommata ist hier leider nicht möglich. Nach der Eingabe drücken Sie <RETURN>. Im Normalfall übernehmen Sie aber einfach das Sternchen mit <RETURN>, dann werden sämtliche Files selektiert.

Die Eingabe eines Dollarzeichens ($) bewirkt, daß das Directory der eingelegten Diskette auf dem Bildschirm ausgegeben wird. Ca. alle 22 Zeilen hält die Ausgabe an und kann durch Tastendruck fortgesetzt werden. Die Taste <RUN STOP> ermöglicht den vorzeitigen Abbruch.

Das Programm reagiert übrigens gelegentlich etwas anfällig auf Directory-Manipulationen, wie etwa CHR$(160) oder Nullbytes im Filenamen, oder Kommentare im Directory, wie man sie mit dem »Disc Wizard« erzeugen kann. Daher sollten Sie möglichst nur ganz normal gespeicherte Files durchgehen lassen, dann kann nichts passieren. Das Programm ist natürlich so programmiert, daß Abstürze unmöglich sind, wenn Sie sich an die Anweisungen auf dem Bildschirm halten. Diskettenfehler werden grundsätzlich abgefangen und angezeigt. Dennoch können eben Files, deren Einträge im Directory manipuliert sind, nicht in allen Fällen getestet werden. Aufpassen sollten Sie auch bei Files, bei denen im Directory als Starttrack und -sektor die Null angegeben ist (solche Files erzeugen manche Directory-Manipulatoren etwa bei den Trennstrichen): diese Files können weder von »TSS«, noch vom

normalen DOS geladen werden.

Nach der Eingabe dieser Vorwahl werden nun das Directory wie gewünscht geladen und die Filenamen eingelesen. Treten hier Diskettenfehler auf, werden diese angezeigt und müssen mit einem Tastendruck quittiert werden. Danach kann die Operation wiederholt werden. Files, die vor dem Typ ein Sternchen stehen haben (sog. nicht geschlossene Write-Files) können nicht gelesen werden, hier erscheint eine blinkende Meldung, die durch Tastendruck bestätigt wird. Das File wird übersprungen. Nach dem Einlesevorgang erscheint die Anzahl der Files, deren Namen bereits gespeichert sind, und die Frage, ob eine weitere Diskette getestet werden soll. Drücken Sie hier auf die Taste <J>, so erscheint die Aufforderung, die Floppy zu wechseln. Daraufhin wird wieder der Directory-Name eingegeben. Somit können auch doppelte Files erkannt werden, die sich unter dem gleichen oder verschiedenen Namen auf mehreren Disketten befinden. Antworten Sie dagegen mit <N>, ist der Vorgang des Namen-Einlesens beendet.

Jetzt prüft das Programm, ob sich mindestens zwei Files im Speicher befinden. Wenn nicht, erscheint eine entsprechende Meldung, denn es hat wohl wenig Sinn, mit einem oder gar keinem File diesen Vergleich durchzuführen. Übrigens dürfen nicht mehr als 250 Files bearbeitet werden, auch eine Überschreitung dieses Maximalwertes erkennt das »TSS«. Mit der Taste <A> können Sie jetzt bei Bedarf abbrechen.

Haben Sie also mit <N> geantwortet, wird jetzt gewählt, nach welchem Verfahren geprüft werden soll. Wie bereits erwähnt, werden vier Prüfsummen gebildet, außerdem kann anhand der Startadresse und der Filelänge auf Gleichheit geprüft werden. Mit den Tasten <1> bis <6> werden jetzt diese sechs Verfahren ein- oder ausgeblendet, jede erdenkliche Kombination (außer: alles sechs ausgeschaltet) ist erlaubt. Ein Verfahren ist eingeschaltet, wenn das Feld mit der Nummer markiert, also revers unterlegt ist. Die Beschreibung der sechs Verfahren:

1: Das erste Verfahren betrifft die Länge des Files in Bytes. Es wird einfach gezählt, bei Gleichheit ist diese Prüfung erfüllt. Der Zähler der Anzahl Bytes hat eine Breite von 16 Bit, läuft also erst bei Files ab 258 Blocks über und beginnt wieder bei Null:

```
      INC SUMME1:BNE LABEL:INC SUMME1+1
LABEL ...
```

2: Eine einfache Prüfsumme: Es werden alle Bytes des Files addiert. Bei Überträgen von einem Byte wird noch eins dazuaddiert. Der Algorithmus sieht folgendermaßen aus:

```
LDA BYTE:CLC:ADC SUMME2:ADC #0:STA SUMME2
```

3: Hier werden alle Bytes des Files EOR-miteinander verknüpft:

```
LDA BYTE:EOR SUMME3:STA SUMME3
```

4: Während bei den Algorithmen 2 und 3 Dreher nicht erkannt werden (diese errechnen etwa bei den aufeinanderfolgenden Zahlen 45, 34 und 63 immer die selbe Summe, egal, in welcher Reihenfolge diese Zahlen im Programm aufeinanderfolgen), berücksichtigt die Methode 4 auch die Stellung eines Bytes im Programmtext. Dazu wird zunächst die bisherige Summe um ein Bit nach links rotiert, Überträge wandern rechts wieder in das Byte hinein, und zum Ergebnis wird dann das jeweilige Programmbyte addiert:

```
LDA SUMME4:ASL:ADC #0:ADC BYTE:STA SUMME4
```

5: Der fünfte Algorithmus ist eine Mischung aller vorangegangenen Methoden. Die bisherige Anzahl der Bytes im File (nur das Lowbyte) wird mit dem gelesenen Byte EOR-verknüpft. Vom Ergebnis subtrahiert der Computer die bisherige fünfte Prüfsumme, rotiert das Ergebnis nach links und speichert es als neue fünfte Prüfsumme ab:

```
LDA SUMME1:EOR BYTE:CLC:SBC SUMME5:ASL:ADC #0:STA SUMME5
```

6: Die sechste Methode, die nach dem Programmstart abgeschaltet ist (das Feld mit der »6« ist nicht markiert), vergleicht die Startadressen der Files.

Bei den obigen Auszügen der Algorithmen bedeutet BYTE das momentan gelesene Programmbyte, SUMME1 bis SUMME5 sind die gespeicherten Prüfsummen, die vor dem Scannen des Files alle auf Null gesetzt werden. Die obigen Programmausschnitte werden für jedes Bytes des Files neu durchlaufen, übrigens unabhängig davon, ob das jeweilige Verfahren eingeschaltet ist oder nicht.

Wie gesagt, es können mehrere oder alle Verfahren miteinander kombiniert werden. Schalten Sie etwa nur das zweite und das sechste ein, so bekommen Sie alle Files mit gleicher Startadresse als Zwillinge gemeldet, deren EOR-Verknüpfung den selben Wert ergibt: Die Verfahren 1, 3, 4 und 5 werden dann nicht berücksichtigt. Im Normalfall ist es aber am sichersten, die Voreinstellung zu übernehmen, die das Programm anbietet, nämlich die Verfahren 1 bis 5 einzuschalten und auf das sechste Verfahren (Startadresse) zu verzichten. Dann werden mit 100 prozentiger Treffer- und praktisch Null-prozentiger Fehlerquote alle wirklich identischen Files unabhängig von ihrer Startadresse angezeigt.

Die Auswahl beenden Sie mit der Taste <RETURN>. Drücken Sie <SHIFT RETURN>, wird in den Automatikmodus geschaltet. Ist dieser aktivert, wartet das Programm im Folgenden nicht mehr auf Eingaben des Benutzers, sondern gibt automatisch alle möglichen Daten auf dem Drucker aus. Dieser Modus kann nicht gewählt werden, wenn Sie mehr als eine Diskette gewählt haben, da dann zum Diskwechsel eine Taste gedrückt werden muß. In diesem Fall reagiert der Computer nicht auf <SHIFT RETURN>.

Jetzt werden alle gewählten Files gescannt. Dies kann je nach Filelänge einige Zeit dauern. Diskettenfehler werden auch jetzt angezeigt, führen allerdings in dieser Phase nicht zum Abbruch, sondern zu der Alternative »Weiter« (Taste <W>) oder »Abbruch« (Taste <A>). Wird die Taste <W> betätigt, fährt das Programm mit dem nächsten File fort, das übersprungene Files wird als gesperrt gekennzeichnet und nicht für die Zwillingsprüfung verwendet.

Immer, wenn Sie beim Einlesen der Directories die Diskette wechselten, erscheint an dieser Stelle jetzt die Aufforderung, eben solches wieder zu tun. Achten Sie peinlich genau auf die Reihenfolge der Diskette, wird versehentlich eine falsche Diskette eingelegt, zeigt das Programm eine »FILE NOT FOUND«-Meldung an. Legen Sie dann erst die richtige Floppy ein, kann das betroffene File nicht mehr gescannt werden und erhält den Sperrvermerk.

Nachdem alle Files gescannt sind, erscheint ein kleines Menü. Sie haben jetzt die Möglichkeit, alle Files in einer Gesamtübersicht auf dem

Bildschirm (Taste <S>) oder dem Drucker (Taste <D>) auszugeben. In dieser Übersicht folgen hinter dem Namen die Startadresse, die Länge in Byte, die um eins erhöhte errechnete Endadresse und die vier Prüfsummen (alle Angaben hexadezimal). Wird die Übersicht auf dem Schirm ausgegeben, fehlen aus Platzgründen die Prüfsummen. Außerdem müssen Sie hier nach der Ausgabe und ca. alle 22 Zeilen eine Taste drücken. Gesperrte Files (bei denen Fehler beim Scanning auftraten) erkennen Sie schon in der Übersicht an einem entsprechenden Vermerk an Stelle der Startadresse und Länge. Ist der Automatikmodus eingeschaltet, gibt der C 64 die Liste automatisch auf dem Drucker aus.

Nach der Ausgabe der Liste oder dem Überspringen mit der Taste <N> kommen Sie jetzt in das Menü des eigentlichen Zwillingsfinders. Dieser kann sein Ergebnis wiederum auf dem Drucker (Taste <D> oder wenn der Auto-Modus eingeschaltet ist) oder Schirm (Taste <S>) ausgeben. Weiter besteht hier die Möglichkeit, mit der Taste <A> abzubrechen, die Taste <Z> führt zurück in die Gesamtübersicht, und die Taste <N> startet das Utility neu.

Bei Anwahl der Twin-Find Funktion vergleicht der Computer jede Prüfsumme mit allen anderen gespeicherten Prüfsummen, oder, besser gesagt, nur diejenigen, die Sie bei der Verfahrensauswahl eingeschaltet haben. Werden Übereinstimmungen festgestellt, gibt der Computer dies aus. Am Ende der Liste erscheint die Gesamtzahl der geprüften Files und die Anzahl der Unikate, also jener Files, die nur genau einmal auftauchten. In der Überschrift dieser Liste werden auch die Verfahren aufgeführt, nach denen sie zusammengestellt wurde. Bei der Bildschirmausgabe soll auch hier ca. alle 22 Zeilen eine Taste gedrückt werden, um ein »Durchscrollen« der Liste zu vermeiden.

Danach befindet sich das Programm wieder im »Twin-Menü«, ein eventuell aktivierter Automatik-Modus wird abgeschaltet.

Sie sehen, das »TSS« ist ein komfortables, vielseitiges und praktisches Utility, das Sie sicherlich gern einsetzen werden. Im folgenden noch die Speicherbelegung (gültig für die Version 2.1, hexadezimal):

0002-0003	temporär
0004-0005	Zeiger für $0e3e
0006	Flag: Automatik
00a6	Modus (gewählte Verfahren)
00a7	momentan bearbeitetes File
00a8	File, mit dem dieses momentan verglichen wird
00a9	Devicenummer für Ausgabe der Liste
00aa	Speicher für X-Register
00ab	Stackpointerspeicher
00b0	Highbyte für $0fc4
00b1	Flag: Hilfsmenü bei Diskfehler
00b2-00b3	Endadresse
00b4	Zeilennummer für Scrolling
00b5-00b6	Speicher für $0002/3
00f7-00f8	Zeiger auf Datenbereich/Hilfsspeicher für Directory
00f9	Anzahl der Files
00fa	gelesenes File-Byte
00fb	Anzahl der Disketten
00fc	Anzahl der Duplikate eines Files
00fd	Anzahl der Duplikatsgruppen
00fe	Anzahl Dubletten insgesamt
02c0-02c7	Prüfsummen des aktuellen Files

0334-0339	Zwischenspeicher für Berechnung der Prüfsummen
0801-08d0	Basic-Lader
08d1-16ac	Programm »TSS 2.1«
08d1-08d3	Sprung zur Hauptroutine bei $1118
08d4-0e10	Texte
0e11-0e25	Filenamen für Directory und Fehlerkanal
0e26-0e2d	Zweierpotenzen
0e2f-0e3d	Hexzahlen
0e3e-1117	Hilfsroutinen
0e3e	Routine: String ausgeben
0e54	auf Taste warten
0e5f	Diskfehler
0e97	Programmabbruch
0fc4	berechnet Zeiger auf Filenamen und Datenbereich
0fff	Verfahren ausgeben
1021	Duplikate ausgeben
10a1	Directory zeigen
1118-16ac	Hauptprogramm
1118	Start Hauptprogramm
118d	Directory lesen
1252	Verfahren wählen
12d3	Files scannen
134a	Prüfsummen bilden
13b9	Menü für Gesamtübersicht
13e7	Gesamtübersicht ausgeben
150a	Menü für Duplikat-Suche
154c	Duplikate suchen
1684	Endadresse ausgeben
1697	Bildschirm-Scroll-Stop
16ac	letztes Programmbyte
6e00-6eff	Flags: File bereits mit anderen verglichen
6f00-6fff	Flags: File gesperrt
7000-8099	Speicher für 250 Filenamen
8100-88d0	Speicher für 250 mal 6 Prüfsummen (je acht Bytes)
8e00-8eff	Tabelle mit den Nummern der Kopien eines Programmes
8f00-8fff	Filenummern mit Diskettenwechsel

Anleitung zum Programm »Verify 2 Files« Version 1.2: Hoch mit dem kleinen Unterschied!

Fast gleich und doch verschieden. Zur genauen Untersuchung der Unterschiede zweier verschiedener Datenfiles eignet sich der Verify-Befehl des Basic nicht. Er kann ja nur »ja« oder »nein« sagen. Ein gutes und einfach zu bedienendes Utility muß her. Eines wie »Verify 2 Files«.

Jeder Commodore 64-Anwender kennt das: Man hat zwei Files, die zwar wahrscheinlich identisch sind, aber eben vielleicht doch nicht. Handelt es sich um zwei Basicprogramme, kann man sich ggf. mit dem Tool »Line-Verify« behelfen. Hat man es aber mit Assemblerprogrammen, Grafikbildern, Zeichensätzen oder anderen Datenfiles zu tun, kann man unter Umständen per Verify-Befehl feststellen, ob es Unterschiede gibt. An welchen Speicherzellen diese stehen, wie viele Unterschiede es sind und welchen

Inhalt die beiden Files an dieser Stelle jeweils haben, bleibt verborgen. Files, die im RAM unter dem ROM liegen, oder in der Zeropage, oder zwei Files mit verschiedenen Startadressen oder unterschiedlicher Länge können mit VERIFY überhaupt nicht richtig verglichen werden.

In allen diese Fällen schafft das Programm »VERIFY 2 FILES« in der vorliegenden Version 1.2 Abhilfe. Das aus Gründen des Komforts und der Geschwindigkeit in reiner Maschinensprache verfaßte Utility (Assemblerkenntnisse sind zu Anwendung nicht nötig) wird wie ein normales Basicprogramm mit

```
LOAD "VERIFY 2 FILES",8
```

geladen und mit

```
RUN
```

gestartet. Auf dem Bildschirm erscheint jetzt die Frage nach dem ersten Programmnamen. Geben Sie den Namen des ersten zu vergleichenden Files, ggf. durch Joker vereinfacht, ein und drücken <RETURN>. Danach wird auf die selbe Art und Weise der Name des zweiten Files eingegeben.

Legen Sie die Diskette ein, auf der sich das erste File findet, und drücken irgendeine Taste. Die Datei wird, unabhängig von ihrer wahren Startadresse, in einen Puffer in den Computerspeicher eingelesen. Als kleine Zugabe ermittelt der Rechner die echten Start- und Endadressen dieses Files und gibt sie aus. Während des Ladens flimmert der Bildschirmrahmen als Anzeige dafür, daß der Computer nicht abgestürzt ist (was bei diesem Tool gar nicht denkbar ist). Für das erste File steht ein Puffer von Speicherzelle 9369 bis 53247 zur Verfügung, also ungefähr 48 KByte oder 172 Blocks. Sollten Sie versuchen, ein längeres Programm zu laden, läuft der Speicher über. Der Computer gibt eine entsprechende Meldung aus und bricht die Bearbeitung ab. Übrigens werden auch Diskettenfehler automatisch erkannt und ausgegeben.

Steht das erste File komplett im Speicher, kann es mit der zweiten Datei verglichen werden. Legen Sie die entsprechende Diskette ein und drücken eine Taste. Die Startadresse des zweiten Files erscheint.

In den untersten drei Bildschirmzeilen können Sie nun die Bearbeitung mitverfolgen. In der drittletzten Zeile wird der Inhalt des ersten Files durchgescrollt, darunter finden Sie die Bytes aus dem zweiten Programm. Ganz unten ist die »Unterschiede-Zeile« untergebracht (Zeichen »D:«). Sie ist normalerweise leer, unterscheiden sich jedoch das erste und das zweite File, scrollt hier ein reverser Block durch.

Da die Bearbeitung in Assembler erfolgt, werden die beiden Files sehr schnell verglichen. Das menschliche Auge kommt kaum mit, nur hin und wieder ist ein ganz kurzer Stop erkennbar, wenn die Floppy einen neuen Sektor liest. Immer, wenn ein Unterschied auftritt, wird das waagerechte Scrolling kurze Zeit sehr langsam und dann allmählich wieder schneller. Diesen Effekt können Sie aber auch durch Druck auf die Shift-Taste erreichen: Drücken Sie, wird das Scrolling immer langsamer, um erst dann wieder zu beschleunigen, wenn Sie die Taste loslassen.

Ist auch das zweite File beendet, endet die Ausgabe. Die Endadresse dieser Datei sowie die Anzahl der erkannten Unterschiede wird noch ausgegeben, dann befindet sich der Computer wieder im Direktmodus. Das Utility kann bei Bedarf durch RUN wieder gestartet werden, oder, wenn der Header (der bei

LIST erscheint) gelöscht wurde, durch Eingabe von SYS 8192.

Dieses nur fünf Blocks kurze Utility eignet sich übrigens auch hervorragend zum Vergleichen zweier unterschiedlich langer Programme, etwa, wenn die Anfänge der beiden Dateien übereinstimmen. Treten zu viele Unterschiede auf, ist davon allerdings abzuraten, da die Ausgabe dann wegen der automatischen Verlangsamung bei Unterschieden sehr lange dauert. Im Normalfall wird man also zum Beispiel zwei unterschiedliche Versionen einer Datei vergleichen, bei der keine Verschiebungen stattfinden (also zum Beispiel Datenfiles oder Maschinenprogramm, die nachträglich per Monitor stellenweise modifiziert wurden).

Sag niemals nie! - »Unscratch« stellt gelöschte Dateien wieder her

Auf dem Computer sind die wenigsten Vorgänge endgültig, auch wenn es oft so aussieht. Die meisten lassen sich nachträglich rückgängig machen. Das Programm »Unscratch« kommt zur Anwendung, wenn Sie versehentlich ein File mit dem Scratch-Befehl von Diskette gelöscht haben: Es kann zurückgeholt werden!

Kennen Sie das Gefühl? Man hat eben völlig auf Versehen eine sehr wichtige Datei mittels Scratch von Diskette gelöscht. Trockenes Schlucken. Eigentlich sollte ja ein anderes File daran glauben, aber wie es in der Hektik und kurz nach Mitternacht nun mal ist... Naja, die Datei ist ja wohl verloren.

So soll und muß es in Zukunft nicht mehr sein. Besitzer eines 1541-Laufwerks können zum einen Dateien vorsorglich vor dem Scratch-Befehl schützen, beispielsweise mit Hilfe des im 64'er Sonderheft 33 veröffentlichten Programmes »File-Lock«. Die andere Möglichkeit wirkt, wenn es passiert ist. »Unscratch« stellt gelöschte Dateien wieder her. Beim Löschen, beispielsweise über die Befehlsfolge

```
OPEN 15,8,15
PRINT#15,"S:DATEINAME"
CLOSE 15
```

wird das File nämlich nicht wirklich gelöscht, sondern nur im Directory als gelöscht gekennzeichnet. Auf der Diskette ist es aber noch vorhanden, und zwar so lange, bis Sie weitere Files darauf speichern. Haben Sie also ein File versehentlich gelöscht, sollten Sie sofort nachdem Sie Ihren Lapsus bemerkt haben »Unscratch« laden. Haben Sie noch keine neue Datei auf der Diskette gespeichert, können Sie davon ausgehen, daß das versehentlich gelöschte File ohne Verluste wiederhergestellt werden kann. Obwohl »Unscratch« aus Geschwindigkeits- und Komfortgründen vollständig in stark optimierter Maschinensprache geschrieben wurde, müssen Sie diese Sprache nicht kennen und auch sonst keinerlei Programmier-Kenntnisse mitbringen, um mit dem Hilfsprogramm umgehen zu können. Es kann wie ein normales Basicprogramm geladen, gestartet und ggf. kopiert werden. Laden Sie es also mit

LOAD "UNSCRATCH",8

Der LIST-Befehl bringt hier keine vernünftigen Befehle zutage, starten Sie also sogleich mit RUN. Auf dem Bildschirm erscheint das Titelbild und die Aufforderung, die Diskette einzulegen, auf der sich die gelöschte Datei befindet. Tun Sie dies und drücken eine Taste. Das Programm liest nun intern das Directory dieser Diskette ein. Gelingt dies nicht, erscheint eine entsprechende Meldung, und das Programm kann nach einem Tastendruck neu gestartet werden. Übrigens werden im gesamten Programm sämtliche Diskfehler abgefangen und angezeigt. Wichtig für Besitzer mehrerer Laufwerke: Nach dem Start stellt sich »Unscratch« auf das Laufwerk ein, das Sie zuletzt angesprochen haben, im Normalfall also das Drive, von dem »Unscratch« geladen wurde. Im Zweifelsfalle setzt das Programm die Geräteadresse 8 (Laufwerk 0) ein.

Der C 64 durchsucht nun das Directory nach als gelöscht gekennzeichneten Files. Wurde ein solches gefunden, erscheint der Filename auf dem Schirm und Sie werden gefragt, ob dieses File wiederhergestellt werden soll. Antworten Sie mit <J> für Ja oder <N> für Nein. An dieser Stelle kann auch abgebrochen werden, dazu ist <A> zu betätigen. Haben Sie sich für die Wiederherstellung entschieden, erscheint als nächstes die Frage nach dem ursprünglichen Filetyp. Sie müssen dem Computer jetzt mitteilen, um was für eine Dateiart es sich handelte. Im Normalfall antworten Sie hier mit der P-Taste, die Datei wird dann als Programmfile markiert. Sollte es sich um eine Datei gehandelt haben, die ein eigenes Programm angelegt hat, kann auch <S> richtig sein, diese Taste erzeugt ein sequentielles File. Andere Möglichkeiten: <U> (»USR«) erzeugt eine vom Anwender definierte Datei (User-File), <R> ergibt das RELative File. Mit <A> kann hier der Vorgang abgebrochen werden, das Programm fährt dann mit dem nächsten File fort.

Ansonsten ersetzt »Unscratch« die Markierung »File gelöscht« im Directory durch den von Ihnen gewählten ursprünglichen Filetyp. Die Datei ist somit wiederhergestellt. Auf diese Weise geht der Computer nun alle gelöschten Dateien auf dieser Diskette durch und fragt jedesmal, ob sie wiederhergestellt werden soll. Sie antworten brav jedes Mal mit <J> oder <N>. Zum Schluß muß, falls mindestens ein File repariert wurde, noch die Organisationsstruktur der Diskette wiederhergestellt werden, das heißt, es werden die Bereiche, auf denen die reparierte Datei steht, als »belegt« gekennzeichnet. Dies kann besonders bei gefüllten Disketten durchaus einige Sekunden oder Minuten dauern (es wird ein »Validate« ausgeführt). Danach ist das Programm fertig, und Sie können wieder wie gewohnt mit Ihrer Datei arbeiten.

Übrigens besteht kein Grund zur Unruhe, wenn Sie sich bei der Wahl des Filetyps vertan haben. Löschen Sie einfach die reparierte Datei sofort wieder mit Scratch und rufen sodann noch einmal »Unscratch« auf, geben diesmal aber den richtigen Typ ein. Beachten Sie bitte auch, daß »Unscratch« nur dann korrekt arbeiten kann, wenn Sie nach dem versehentlichen Löschen der Datei noch keine weiteren Files auf dieser Diskette geschrieben haben. Andernfalls könnte es passiert sein (muß aber nicht, einen Versuch ist es zumindest wert), daß die neuen Files das alte, versehentlich gelöschte überschrieben haben. Nach der vollständigen Reparatur der Diskette können Sie diese selbstverständlich wieder ohne Einschränkungen nutzen.

Wir wünschen Ihnen viel Erfolg beim Retten versehentlich gelöschter Dateien mit »Unscratch«.

Top Secret - Daten sicher geschützt

Datenschutz auf dem Computer ist in! Schließen auch Sie Ihre geheimen Files vor neugierigen oder unbefugten Blicken sicher weg. »Top Secret«, ein sehr komfortables Maschinensprache-Utility codiert Files aller Art direkt auf Diskette. Nicht einmal ein Experte mit Diskmonitor wird ohne Kenntnis des Paßworts an die Daten gelangen. Dabei findet ein Chiffrier-Algorithmus Verwendung, der sonst etwa von Geheimdiensten eingesetzt wird. Das ist es: Dateien besser verschlüsseln als der BND!

Haben Sie eine spezielle Programmiertechnik, die vor fremden Blicken geschützt werden soll? Oder enthält Ihr Programm DATA-Zeile, die vor fremden Blicken sicher sein sollen? Verwalten Sie Personendaten und wollen die Bestimmungen des Bundesdatenschutzgesetzes (BDSG) exakt einhalten? Oder stellen Sie sich vor, Sie haben eine Diskette randvoll mit Programmen, die Sie an verschiedene Bekannte weitergeben möchten, von denen aber nicht jeder alle Programme erhalten soll, sondern nur einen für ihn bestimmten Teil. Schützen Sie alle Programme und geben jedem File ein individuelles Paßwort. Dann können Sie die ganze Disk vervielfältigen und müssen Ihren Freunden nur jeweils die für sie bestimmten Codewörter mitteilen.

Das Programm »Top Secret« codiert sequentielle Dateien auf Diskette, also PRG, SEQ und USR-Files. Das Programm läßt sich danach zwar noch laden, es erscheint aber nur Unsinn. Nur mit Hilfe von Top Secret lassen sich die Daten wieder rekonstruieren. Dabei kann für jede Datei ein individuelles Codewort vereinbart werden, ohne das kein Zugriff möglich ist. Sichere Verschlüsselungs-Mechanismen garantieren absolute Datensicherheit selbst vor Profis, denen Teile des Originals bekannt sind. Obwohl das Programm aus Gründen des Komforts und der Geschwindigkeit vollständig in Maschinensprache geschrieben ist, benötigen Sie keinerlei Assemblerkenntnisse, um damit arbeiten zu können. Das sichere und leicht zu bedienende Utility kann wie ein Basicprogramm geladen, gestartet und ggf. kopiert werden. Sie sollten nur niemals den Fehler machen und Top Secret mit sich selbst codieren, wenn Sie keine Kopie mehr davon haben. Dann ist nämlich keine Rückgängigmachung mehr möglich!

Laden Sie das Programm mit

```
LOAD "TOP SECRET",8
```

Der Start erfolgt mit RUN. Legen Sie die nicht schreibgeschützte Diskette mit der zu schützenden oder freizugebenden Datei ein, und betätigen eine Taste. Aus technischen Gründen können keine GEOS-Disketten verarbeitet werden, das Programm erkennt diesen Fall automatisch und bricht mit einer entsprechenden Meldung ab. Weitere Einschränkung: Dateien vom Typ DEL und REL, die in der Praxis kaum vorkommen, lassen sich aufgrund ihrer komplizierten Struktur ebenfalls nicht chiffrieren. Ausgenommen sind außerdem »open write-files«, also Dateien mit einem Stern vor dem Filetyp im Directory sowie Files, die auf dem Track 18 lagern. Dazu gehören insbesondere die Trennstriche, die von Programmen wie dem Disc-Wizzard erzeugt werden. Da diese Files allerdings keine Daten enthalten, wäre ein

Verschlüsseln ziemlich sinnlos.

Nach dem Einlegen der Diskette darf die Laufwerksklappe bis zum Erlöschen der roten Leuchtdiode nicht mehr geöffnet werden. Es erscheint eine Auflistung der auf dieser Diskette gespeicherten in Frage kommenden Dateien. Angezeigt werden Filename und -typ. Hinter dem Typ weist ein Schrägstrich »/« auf eine normal und ein Prozentsymbol »%« auf eine nach dem Delta-Verfahren (siehe unten) chiffrierte Datei hin. Bei jedem gemeldeten File können Sie mit den Tasten <J>, <N> und <A> wählen: <J> selektiert die Datei. Ist sie noch ungeschützt, verzweigt das Programm in die Codier-Routine, bei einer bereits chiffrierten Datei wird das Entsichern eingeleitet. Es ist also nicht möglich, eine bereits codierte Datei nochmals zu verschlüsseln, oder ein File zweimal hintereinander zu dechiffrieren. Antworten Sie mit <N>, geht es mit der nächsten Datei weiter. Bei <A> wie »Abbruch« schließlich wird das Programm vorzeitig beendet.

Haben Sie sich für ein File entschieden, betätigen Sie also <J>. Sowohl beim Codieren wie auch beim Decodieren ist jetzt ein maximal 14 Zeichen umfassendes Paßwort einzugeben. Beim Codieren wählen Sie es beliebig, bei Decodieren ist es als Sicherheitsabfrage wieder einzugeben. Die Tasten <Space>, <Ausrufezeichen> und <Anführungszeichen> sind gesperrt, ansonsten stehen alle alphanumerischen Zeichen (Buchstaben, Ziffern, Satzzeichen) zur Verfügung. Die Eingabe kann mit <DEL> korrigiert werden und wird mit <RETURN> abgeschlossen. Geben Sie nichts ein und drücken nur die RETURN-Taste, geht es in der Auswahl mit dem nächsten File weiter, das vorher gewählte wird dann nicht bearbeitet.

Eine mit Top Secret geschützte Datei kann ohne das Codewort nicht, auch nicht mit Hilfe von Tricks, wiederhergestellt werden. Geben Sie also eine Zeichenfolge ein, die Sie sich leicht merken können, zum Beispiel den Namen von Freundin/Freund, Ihr Geburtsdatum oder dergleichen mehr. Sorgen Sie dafür, daß es nicht zu offensichtlich ist und erraten werden könnte. Wenn Sie eine geschriebene Liste all Ihrer Kennwörter anlegen, bewahren Sie sie an einem sicheren Ort auf. Am besten verwenden Sie für all Ihre Sicherheitsfiles das selbe Paßwort, dann kommt keine Verwirrung auf. Die Eingabe eines fehlerhaften Codeworts wird beim Decodieren mit einer entsprechenden Meldung quittiert, geben Sie dann nochmals das korrekte Paßwort ein.

Achtung: Aus technischen Gründen kann es rein theoretisch in äußerst seltenen Fällen vorkommen, daß der Computer ein absichtlich falsches Paßwort annimmt und dann versucht, damit das File zu decodieren. In diesem Fall wird nur Unsinn erzeugt, die Datei ist dann rettungslos verloren. Grund: Das Paßwort wird nicht Zeichen für Zeichen auf Diskette gespeichert, sondern nur seine Länge und eine Art Quersumme. Hat nun ein falsches Kennwort die selben Daten, akzeptiert das Programm es als korrekt. Da beim Codieren die zu verschlüsselnde Datei mit den Zeichen des Codeworts verknüpft wird und der Decodiervorgang nicht rückgängig gemacht werden kann, hätte ein falsches Paßwort fatale Auswirkungen. Die Routine ist aber so programmiert, daß zum Beispiel Tippfehler wie »Dreher« zuverlässig erkannt werden. Die Einschränkung wirkt auch sonst nicht schwer, da nur etwa 0,3 % aller falschen, von der Länge aber korrekten Paßwörter vom Programm »gefressen« werden. Nur durch diese Technik kann verhindert werden, daß ein Experte durch Auslesen der gesicherten Diskette das Paßwort ermitteln kann. Am sichersten freilich ist es, eine ungeschützte Kopie des Programms auf Band oder Disk an einem sicheren Ort zu verwahren.

Beim Codieren erscheint nun die Abfrage, ob das Delta-Verfahren zu verwenden

ist. Bei diesem relativ kompliziert arbeitenden Verfahren handelt es sich um eine sehr sichere Methode, die auch bei Kenntnis einiger Teile des Originals nicht zu knacken ist und darüber hinaus die chiffrierte Datei nicht verlängert. Dieses Verfahren beruht auf einem Vergleich der im File aufeinanderfolgenden Bits und wurde in Ausgabe 6/90 (Seite 62) des 64'er-Magazins genau besprochen. Übrigens bedienen sich auch viele Geheimdienste bei der Informationsübermittlung des Delta-Verfahrens, das nur den Nachteil hat, daß die Codierung etwas länger dauert. Antworten Sie auf die Frage mit <J>, wird die Datei auch nach dem Delta-Verfahren behandelt. Bei <N> unterbleibt dies, und es wird »nur« der Inhalt der Datei auf geheimnisvolle Weise mit den Zeichen des Paßworts verknüpft (auch bei Eingabe von <J>, dann allerdings eben zusätzlich mit Delta-Codierung).

Beim Decodieren merkt das Programm vollautomatisch, ob die Datei im Delta-Verfahren verschlüsselt wurde oder nicht, und berücksichtigt dies. Eine besondere Eingabe ist nicht erforderlich.

Die Datei wird jetzt zunächst im Directory als geschützt oder wieder decodiert gekennzeichnet. Diese Markierung können Sie auf herkömmliche Weise nicht sichtbar machen, sie erscheint nur in Form des Schrägstrichs bzw. Prozentzeichens bei der Auflistung in Top Secret.

Dann erfolgt der Zugriff auf sämtliche Bereiche der Diskette, die vom File belegt sind. Alle werden verschlüsselt bzw. wieder zurückgerechnet. Gerade bei längeren Dateien kann dies durchaus etwas dauern, im allgemeinen ist etwa die doppelte Ladezeit für den gesamten Vorgang anzusetzen. Beispiel: Eine Datei mit 122 Blocks wird mit Delta in 4:12 Minuten codiert und in 4:38 wieder freigegeben, ohne Delta-Verfahren dauert's jeweils 3:27. Wie in allen anderen Phasen auch werden hier Diskettenfehler erkannt und sofort gemeldet. Beseitigen Sie ggf. die Ursache für die Störung und starten durch einen Tastendruck das Programm nochmals.

Noch einige Hinweise zur praktischen Anwendung. Beim Codieren von PRG-Dateien wird auch die in den ersten beiden Bytes gespeicherte Ladeadresse verändert. Versuchen Sie also, eine geschützte Datei absolut (,8,1) an die vorgesehene Stelle in den Speicher zu holen, so wird Ihnen nicht einmal das gelingen. Vorsicht beim Kopieren von geschützten Files: Da Top Secret die Organisation der geschützten Diskette verändert, können chiffrierte Files weder mit File-Kopierprogrammen, noch beispielsweise mit dem Copy-Befehl der Diskettenstation dupliziert werden. Die Kopien lassen sich dann weder laden (weil sie codiert sind) noch decodieren (weil die Information, daß sie chiffriert sind, nicht mitkopiert wurde und daher von Top Secret nicht mehr gefunden wird). Dennoch lassen sich auch gesicherte Dateien ganz normal löschen, auch gibt es sicher keine Probleme beispielsweise beim Validieren. Beim Kopieren mit einem Backup-Programm, das die gesamte Diskseite vervielfältigt, sind keine Probleme zu erwarten.

Zuletzt noch einige kurze Hinweise zur Funktionsweise. Das Programm vermerkt in den normalerweise (außer bei REL- oder GEOS-Dateien) unbenutzen Bytes 22, 23 und 24 des Directory-Fileeintrags, ob und ggf. wie das File codiert wurde. Bit 7 in Byte 22 wird für ein geschütztes File gesetzt, Bit 6 ist 1, wenn das Delta-Verfahren aktiviert wurde. In Byte 23 findet sich eine Prüfsumme über das Codewort. Zusammen mit Byte 24, in dem codiert die Länge des Paßworts erfaßt wurde, erfolgt so beim Entschlüsseln die Prüfung, ob die Eingabe korrekt ist. Die Codierung der Filedaten selbst erfolgt per Direktzugriff (U1 und U2-Befehle) auf die vom File belegten Sektoren. Für jeden Sektor wird das erste Byte mit dem ersten Byte des Paßworts verknüpft, das zweite Byte mit dem zweiten Zeichen und so weiter bis zum Ende des

Codeworts. Danach geht es mit dem ersten Byte des Kennworts von vorn los. Außerdem erfolgen Bit-Rotationen, um den Schutz zu verbessern. Falls gewünscht, wird danach noch die Delta-Codierung durchgeführt, für die im Programm eine eigene Routine vorhanden ist.

Den Bits auf der Spur: Der »Disk Spy«

Rund, schwarz und geheimnisvoll sind die Disketten für das 1541-Laufwerk. Zumindest die letzte Eigenschaft verlieren sie aber ganz schnell, wenn Sie mit einem Diskettenmonitor rangehen. Der einfach zu bedienende Disk Spy bietet wichtige Funktionen zur Manipulation von Daten direkt auf der Magnetscheibe. Sogar ein Disassembler ist enthalten.

Bei diesem Diskmonitor handelt es sich um eine Weiterentwicklung des von Matthias Boeing in RUN 2/86 vorgestellten Diskmonitors, allerdings in völlig neu programmierter Fassung. Unsere Version ist um einiges bedienungsfreundlicher, alte Funktionen wurden stark verbessert, teilweise wurden neue hinzugefügt, so beispielsweise der umfangreiche Hilfsschirm, der mit dem Kommando »?« aufgerufen werden kann. Dieser Monitor hat 22 Befehle, die Ihnen das Arbeiten mit der Diskstation einfacher machen. Sie werden weiter unten aufgelistet.

Doch vorher noch einige Worte zur Bedienung. Der Monitor wird mit dem Befehl

```
LOAD "DISK SPY",8
```

geladen. Dabei geht ein evtl. im Speicher stehendes Basicprogramm verloren. Geben Sie nach dem Laden den Befehl RUN ein. Das Maschinenprogramm wird an seine eigentliche Adresse verlegt (30000) und dort gestartet. Hierauf meldet sich das Programm mit einem hellen Bildschirm, auf dem in dunkler Farbe eine Einschaltmeldung steht. Nun zeigt das Programm den Fehlerkanal der Diskstation an. Hier sollte der Text »00,OK,00,00« erscheinen, wenn nicht, schalten Sie Floppy und Computer ab, und laden erneut. Unter der Meldung hat der Rechner bereits den »?« Befehl angezeigt. Quittieren Sie diesen, indem Sie die RETURN-Taste drücken. Es wird nun eine Liste aller vorhandenen Befehle mit Erklärung angezeigt. Diese Befehle bestehen jeweils aus einem Buchstaben, dahinter teilweise noch Angaben, auch »Parameter« genannt.

Bevor wir Ihnen jetzt die Liste aller Befehle präsentieren, noch die übliche Diskmonitor-Warnung: Um sich mit dem Programm vertraut zu machen, sollte man keine Disk verwenden, auf der sich wichtige Daten befinden. Ein Diskmonitor ist ein sehr mächtiges Werkzeug, und kann bei Fehlbedienung irreparablen Schaden anrichten! Auch wenn dieser Monitor weitgehend gegen falsche Bedienung gesichert ist, beherzigen Sie diesen Rat während der Einarbeitungszeit im Interesse Ihres Disketteninhalts.

An dieser Stelle eine Erklärung wichtiger Begriffe im Zusammenhang mit der 1541-Diskette, in aller Kürze. Genauere Informationen finden Sie in zahlreichen Kursen zu diesem Thema, die auch im 64'er-Magazin immer wieder erscheinen. Die Diskette ist in 35 konzentrische Kreise um den Mittelpunkt,

die »Spuren« oder »Tracks« aufgeteilt. Spur 1 ist ganz außen, Track 35 innen. Auf manchen Disketten sind beispielsweise zu Kopierschutzzwecken auch noch die Spuren 36 bis 42 vorhanden, die von Disk Spy aus technischen Gründen allerdings leider nicht bearbeitet werden können. Je nach Nummer enthält ein solcher Track bis zu 21 Sektoren, die die eigentlichen Daten enthalten. Die Sektorgrenzen verlaufen im Prinzip über die Tracks hinweg strahlenförmig vom Mittelpunkt der Diskette weg. Die folgende Tabelle enthält die Anzahl der Sektoren pro Track:

Spur	Anzahl Sektoren
1 bis 17	je 21
18 bis 24	je 19
25 bis 30	je 18
31 bis 42	je 17

Beispiel: Spur Nr. 21 enthält 19 Sektoren, die von 0 (!) bis 18 durchnumieriert werden. Die unterschiedliche Anzahl der Sektoren pro Spur ist bedingt durch die Verkürzung der Spuren zum Mittelpunkt hin. Ein Sektor speichert 256 Bytes. Addieren Sie die Sektorenzahlen der Tracks 1 bis 35, erhalten Sie den Wert 683, das entspricht den 174848 Bytes (170,75 KB), die eine 1541-Diskette speichern kann. Der Track 18 wird komplett vom Inhaltsverzeichnis der Diskette belegt, zieht man von den 683 Blocks die 19 Blocks der Spur 18 ab, kommt man auf die 664 Blocks, die nach dem Formatieren »FREE« sind.

Der wichtigste Block einer Diskette findet sich auf Track 18, Sektor 0 (vor dem Inhaltsverzeichnis) und enthält die »Block Availability Map«, kurz BAM. Es handelt sich um eine Tabelle, in der für jeden der 664 Sektoren einer Diskette verzeichnet ist, ob er von Daten belegt oder noch verfügbar, also frei ist. Aufgrund dieser Informationen errechnet das Laufwerk die »xxx BLOCKS FREE.«, die unter jedem Directory erscheinen. In diesem Block stehen außerdem Name und ID der Diskette. Sollten jetzt noch Begriffe unklar sein, schlagen Sie im Lexikon am Ende des Buches nach.

Nun zur Auflistung der Befehle. Sie geben den Kennbuchstaben ein, wenn nötig und/oder erwünscht, dahinter noch die Angaben, meistens im Hexdezimalsystem. Gibt es mehrere Schreibweisen für einen Befehl, folgen diese vor der jeweiligen Erklärung aufeinander. So ist es etwa beim zweiten Befehl, dem m-Befehl, möglich, nur das m zu schreiben, oder mit einem oder zwei Parametern dahinter. Lassen Sie bei einem solchen Befehl Parameter weg, werden diese vom Programm sinnvoll ergänzt (vgl. Beschreibung).

r tt ss
Der mit TT (Track) und SS (Sektor, beide hexadez.) angegebene Block wird in den Speicher gelesen. Die Befehle m, w, v, f, p, h, d, e und t werden nur ausgeführt, wenn sich bereits ein Block im Speicher befindet.

m
m ab
m ab eb
Der im Speicher stehende Block wird von AB bis EB (hexadezimal, wenn nicht anders angegeben: AB = 0, EB = 255) hexadezimal und im ASCII-Code angezeigt. Durch <STOP> kann die Anzeige abgebrochen werden. Um ein Byte zu ändern, geht man mit den Cursor auf die entsprechende Stelle und ändert sie. Änderungen im ASCII-Code (rechts) sind nicht möglich. Statt der HEX-Bytes kann auch der Großbuchstabe A gefolgt vom Zeichen verwendet werden. Der Code des Zeichens wird dann eingesetzt. Nach Druck auf <RETURN> werden die Änderungen im Speicher fixiert, auf Disk gelangen sie erst durch den w

Befehl. Im ASCII Code werden nicht druckbare Zeichen als ».« augegeben.
Beispiel: Aus

```
]:12  30 31 32 33 34 35 36 37   01234567
```

wird durch Überschreiben:

```
]:12  30 33 Aw A9 34 35 36 37   01234567
```

nach Druck auf <RETURN> werden die Änderungen in den Speicher und die ASCII Anzeige übernommen.

Bei den Laufwerken für den C 64 enthalten die ersten beiden Bytes jedes Sektors die Nummer des Tracks und Sektors, auf dem die Datei fortgesetzt wird. Steht hier als Trackangabe eine Null, ist das ein Hinweis, daß dieser Block der letzte Datenblock des Files ist. An Stelle der Sektorangabe findet sich dann die Anzahl der Bytes, die die Datei auf diesem Block noch belegt. Der Disk-Spion wertet nach dem m-Befehl die beiden Angaben automatisch aus und gibt einen r-Befehl aus, der nur noch mit <RETURN> bestätigt werden müßte, um den nächsten Block zu lesen. So hangelt man sich unkompliziert auch durch größere Files.

w
w tt ss
Der im Speicher stehende Block wird auf die Disk geschrieben. Wird TT und SS weggelassen, werden Track und Sektornummer des letzten Lese- oder Schreibbefehls verwendet.

h aa bb cc
h aa bb Ax
Alle Stellen innerhalb aa und bb, in denen das Byte cc bzw. das ASCII Zeichen x auftritt, werden gelistet. Am Ende wird die Anzahl der Fundstellen hexadezimal ausgegeben. Die Suche kann mit <STOP> abgebrochen werden.

v
Der Block im Computer wird mit Nullen gefüllt.

f aa bb cc
f aa bb Ax
Der Block im Computer von aa bis bb wird mit dem Byte cc (wie üblich hexadezimal) oder dem ASCII Code des Zeichens x gefüllt. Falls die ASCII-Darstellung gewählt wird, muß das A wie auch beim m Befehl mit <SHIFT> eingegeben werden!

p aa text
Der Text wird ab aa in den Block im Computer geschrieben. Er braucht nicht in Anführungszeichen zu stehen, diese werden wie alle anderen Zeichen mitgespeichert.

t
Der zuletzt gelesene Block wird in Form eines w Befehles ausgegeben. Um ihn auszuführen, drücken Sie nur <RETURN>.

A "text"
Der text (in Anführungszeichen) wird als Hexdump ausgegeben. Das A muß geshiftet eingegeben werden, damit keine Verwechslungen mit dem folgenden Kommando auftreten:

a bb bb bb ...
Die Bytes bb bb bb bb (beliebig viele) werden im ASCII-Code ausgegeben (Umkehrfunktion zu A)

@ (Klammeraffe)
Der Fehlerkanal der Floppy wird ausgegeben (@ = Klammeraffe, Taste links neben <Stern>).
@text
Der Text wird als Kommando an die Floppy gesendet. Beachten: Kein Leerzeichen nach dem @! Bsp.: @v (Validate)

@$
Das Inhaltsverzeichnis wird ausgegeben (kein Leerzeichen zwischen @ und $, @ = Klammeraffe)

?
Hilfe-Funktion: Alle Befehle des Monitors werden angezeigt.

<- (Pfeil nach links)
Die Farbwahl des Bildschirms wird geändert. Es sind sieben verschiedene Farbkombinationen gespeichert, die der Reihe nach abgerufen und danach wiederholt werden. Die Taste mit dem Pfeil nach links finden Sie ganz links oben.

c nn
Die Geräteadresse innerhalb der Floppy wird in nn (hex.) geändert und bleibt bis zur erneuten Änderung oder zum Abschalten der Floppy erhalten. Auch spricht der Monitor die Floppy jetzt unter der neuen Nummer an.

n
Der Monitor zeigt an, unter welcher Geräteadresse er die Floppy momentan anspricht.

n nn
Der Monitor spricht die Floppy ab jetzt unter der Nummer nn (hexadezimal, 08 - 0f) an. So kann man mehrere Diskstationen gleichzeitig betreiben. Sinnvoll ist dieser Befehl aber nur, wenn die zweite Floppy beispielsweise mit dem c-Befehl oder hardwaremäßig auf die neue Nummer eingestellt wurde.

$ dz
Die Dezimalzahl dz (0 - 65535) wird hexadezimal angezeigt, Bsp.: $ 12, $ 5856, $ 90.

! nnnn
Die Hexadezimalzahl nnnn (vierstellig) wird dezimal ausgegeben. Beispiele: ! 1342, ! 0002, ! ffd2

d aa bb cc
Der Bereich aa bis cc wird disasembliert, als stünde er ab bb (vierstellig, hexadezimal) im Speicher. Die Branch-Befehle (BEQ, BCC etc) werden entsprechend korrigiert. Der Disasembler akzeptiert auch illegale OP-Codes. »???« sind nicht dechiffrierbare Codes, »CRA« solche, die zum Absturz (CRAsh) führen.

e
e aaaa
Enthält der Block im Computer ein komplettes, also 1 Block langes Programm, so wird es durch den e (Exexute) Befehl in den Speicher des C 64 ab Adresse aaaa (vierstellig, hexadezimal) gelesen. Fehlt aaaa, wird die Adresse angenommen, die in Byte $02 und $03 des Blocks steht. Nach der Ausführung des Befehl wird automatisch ins Basic zurückgesprungen, weil eventuell der Monitor überschrieben wurde. Das geladene Programm kann mit SYS dann

gestartet werden.

x
Der Monitor wird verlassen und ins Basic zurückgesprungen. Er kann mit SYS 30000 wieder gestartet werden, auch nach NEW oder RESET.

s
Der Monitor wird neu gestartet.

Fehlermeldungen:

Wurde ein Befehl nicht verstanden, erscheint ein einfaches Fragezeichen (?). Wurde ein Befehl zwar verstanden, und die Interpretation der Parameter gelang nicht, werden zwei Fragezeichen ausgegeben. War alles korrekt, wird der Befehl ausgeführt und eine neue Eingabe erwartet.

Hinweise zur Praxis des Diskmonitors:

Angenommen, Sie wollen ein mit dem Scratch-Befehl gelöschtes File wieder retten und haben seit dem Löschvorgang noch nicht auf diese Disk geschrieben. Das gelöschte Programm kann dann leicht wieder rekonstruiert werden. Im Folgenden zeigen wir Ihnen den Weg, der hierzu mit dem Disk-Spy gegangen werden müßte. Dieser Vorgang findet automatisiert aber auch beim in dieser Hinsicht komfortableren und sichereren Utility »Unscratch« statt.

Laden Sie den Monitor, starten ihn und lesen den ersten Block der Directory ein: Track 18 (hex. $12), Sektor 1.

r 12 01

Nach kurzer Zeit ist der Block im Computer und könnte geändert werden. Geben Sie den m-Befehl ohne Parameter ein. Es werden nun einige Filenamen durchlaufen (rechts in der ASCII-Anzeige). Wenn Sie den Namen der versehentlich gelöschten Datei wiederfinden, machen Sie weiter wie unten beschrieben, sonst drücken Sie einfach <RETURN> für den am Ende des Hexdumps stehendes r-Befehl, der als Argument gleich die Track und Sektornummer des Folgetracks mitbekommen hat. Steht dort kein r-Befehl, so ist (war) das gelöschte File nicht auf dieser Disk oder wurde schon durch ein neues überschrieben. Steht dort aber noch ein r-Befehl, führen Sie diesen aus, und machen wie oben mit dem m-Befehl ohne weitere Angaben weiter.

Haben Sie den Namen in der ASCII-Anzeige gefunden, drücken Sie schnell die STOP-Taste, um die Ausgabe anzuhalten. Sie bewegen den Cursor nun hoch an den Anfang der Zeile, in der der (Anfang des) Filename(ns) steht. Gehen Sie nun auf das vierte Byte nach dem Doppelpunkt (also das dritte nach dem Doppelleerzeichen). Es muß sich um ein Nullbyte handeln. Überschreiben Sie dieses Byte mit dem Hexbyte »82«, falls das File ein Programmfile war (»81« = SEQ) und drücken die RETURN-Taste. Löschen Sie nun den Bildschirm und drücken Sie RETURN, damit die Klammer (der Prompt »]«) wieder erscheint. Nun geben Sie den w-Befehl ohne Parameter ein und drücken <RETURN>. Die Änderung wird nun auf Disk übertragen. Geben Sie zur Kontrolle den $ Befehl und schauen, ob das File im Directory wieder erscheint. Wenn alles in Ordnung ist, müssen Sie noch ein Validate geben: Der Befehl lautet v. Damit werden die vom File benutzten Blocks in der Belegungstabelle (Block Avilab Map, BAM, siehe oben) wieder als belegt gekennzeichnet. Diese Prozedur kann je nach Diskfüllung sehr lange dauern, die Mühe wird sich aber lohnen. Wie gesagt, üben Sie dies aber vorher an einer Übungsdisk ohne wichtige Programme. Leichter geht's in jedem Fall mit »Unscratch«.

Wir wünschen Ihnen viel Erfolg mit dem Disk-Spy!

Ein Monitor für die Floppy: Anleitung zum D.M.S.

Mit einem normalen Maschinensprachemonitor können Sie nur auf den Computerspeicher zugreifen. Ein Diskettenmonitor liest nur Daten von der Diskette. DMS greift wie ein Maschinensprachemonitor auf den Speicher der 1541 zu. Damit wird die Programmierung des Laufwerks einfach.

Zwei Programmtypen gibt es, die Sie kennen: Der Monitor und der Diskmonitor. Mit einem Monitor kann man sich den Speicher (das RAM und ROM) des C 64, mit einem Diskmonitor die Tracks und Sektoren der Floppy ansehen. Was geschieht nun, wenn man diese beiden Programme miteinander vermischt? Damit meinen wir nicht einen Monitor, der einen Diskmonitor beinhaltet. Wir sprechen vielmehr von einem Monitor, der wie ein »normaler« C 64-Monitor bedient wird, aber Zugriff auf den Speicher des Laufwerks hat! Ein solches Programm ist der D.M.S. (Disk Memory Spy, Diskspeicherspion). Mit leistungsfähigen, einfach zu bedienenden Befehlen rücken Sie der Floppy zu Leibe. Außerdem sind noch sinnvolle Sonderfunktionen integriert, wie das Listen eines Programmes in drei Formaten, das Disassemblieren der FloppyROMprogramme und das Verschieben von Programmen im Floppyspeicher mit Umrechnung der absoluten Adressen.

Sie laden das Programm mit LOAD "DMS",8 und starten es mit RUN. Ein Kurzprogramm verschiebt das Maschinenprogramm an die erwünschte Stelle ab 25000, und startet es. Nun wird ein »I«-Befehl zum Laufwerk geschickt, ein Maschinenprogramm in den Floppyspeicher übertragen (dieses stellt einen verbesserten M-R Befehl (UC) da, da der normale M-R Befehl bei Adresse $CB3E (für Profis) im Floppyrom einen Fehler enthält: der INC Befehl könnte unter bestimmten Umständen das Carryflag setzen).

Nun erscheint eine Meldung, und dann der Eingabeprompt (ein Größerzeichen), das anzeigt, daß das Programm auf eine Eingabe wartet. Jetzt können Sie die Befehle eingeben. Jeder Befehl besteht aus einem Buchstaben mit meistens noch Parametern dahinter. Diese werden als zwei- oder vierstellige hexadezimale Zahlen übergeben. Falls alles korrekt ist, wird der Befehl ausgeführt. Bei einem Fehler wird das Größerzeichen revers gefärbt.

Hier vor der Liste mit allen Befehlen noch einige Tips: Alle Leerzeichen in der Eingabezeile können weggelassen werden. Der Eingabestring wird nach der Eingabe nämlich von einer Routine bearbeitet, die außerhalb von Anführungszeichen alle Spaces, Shiftspaces und Prompts (»>«) aus der Zeile entfernt. Sie können den Befehl M 9800 also auch als »M9800« oder als »M 9 8 0 0« oder als »M >980 >0« eingeben (alle »>«-Zeichen werden überlesen). Innerhalb von Anführungszeichen gilt das nicht. Das zweite Gänsefüßchen kann grundsätzlich weggelassen werden (übrigens auch in BASIC, etwa bei PRINT).

Falls Sie sich mit dem »M«-Befehl Speicherinhalte auf den Bildschirm geholt

haben (auch bei den Befehlen »D«, »N«, »J«, »$«, »#«), können Sie nun sehr einfach Änderungen vornehmen, indem Sie mit dem Cursor an die entsprechende Stelle fahren und das entsprechende Byte überschreiben.

Hier ist eine Liste mit allen Befehlen. Parameter im Klammern können weggelassen werden (siehe Text).

Bedeutung der Zeichen:

Großbuchstabe	Name des Befehles
ad	Adresse (4 stellige Hexzahl)
by	Byte (zweistellige Hexzahl)
text	Beliebiger ASCii-Text ohne Gänsefüßchen
"name"	Filename in Anf.z. (das zweite kann entfallen)
ga	Geräteadresse (hex) 04-FF
puf	Puffernummer in der VC 1541 (hex, 2 stellig, 00-05)
hex	Hexadezimalzahl, 0-4 stellig)
dez	Dezimalzahl (0-65535)
tr	Tracknummer (1-41)
se	Sektornummer (0-21)

Die Befehle: (»@« = Klammeraffe, rechts neben der »P«-Taste)

X	EXIT: Verlassen des Programmes. Neustart, auch nach RESET, mit SYS 25000 (der BASIC Bereich wird entsprechend eingeengt). Wenn Sie hinter den SYS Befehl nach einem Komma noch eine Zahl anhängen, wird diese als neue Deviceadresse gedeutet (SYS 25000,8)
?	HELP: gibt eine Liste der Befehle aus
;	REMARK: alles, was hinter dem »;« steht, wird überlesen. Zweck: Das Programm benutzt dieses Zeichen manchmal, wenn es Listen ausgibt, etwa beim »^« Befehl. Auch kann man hiermit Bemerkungen für eine eventuelle Hardcopy einfügen.
B (tr)	BUMP: erzeugt ein Rattern, um etwa eine Justierung vorzunehmen, wenn der Tonkopf an einer nicht definierten Stelle steht. Falls der Track angegeben ist, wird nach dem Bump auf diesen Track positioniert.
N (ga)	NUMBER: Setzt die Gerätenummer (ga), unter der das Programm das Laufwerk anspricht. Falls ga fehlt, wird eine Nummer ausgegeben, die einfach verändert und/oder mit <RETURN> übernommen werden kann. Diese Adresse kann aber auch mit SYS 25000,ga oder POKE 26263,ga gesetzt werden.
@ (text)	CMD: Falls der Text fehlt: Der Fehlerkanal wird ausgegeben. Manchmal kann es passieren, daß stattdessen nur eine Leerzeile oder Unsinn erscheint. Probieren Sie es dann erneut. Falls der Text mit »$« beginnt, wird die Directory (nach Wahl selektiert, etwa @$op.*) ausgegeben. Abbruch mit <STOP>, Pause mit <SPACE>. Sonst wird der Text auf dem Fehlerkanal zur Floppy geschickt, etwa @S:NAME zum Löschen eines Filets.
*	RESTART: Startet das Programm neu
^ (puf)	LIST JOBS: Der Pfeil nach oben listet die fünf Jobcodes mit den Track- und Sektornummern (dezimal) und einer Erklärung. Hinter dem Pfeil nach oben (»^«) kann der Puffer aber auch einzeln selektiert werden, etwa ^04 für Nr. 4.
$ hex	Die Hexzahl wird ins Dezimale gewandelt und ausgegeben.
# dez	Die Dezimalzahl wird ins Hexadezimale gewandelt
R	RESET DRIVE: Es wird im Laufwerk ein RESET (UJ) ausgelöst. Der Computer wartet (ca. zwei Sekunden), bis die Floppy ihren Speicher gelöscht und sich initialisiert hat.

M (ad) MEMORY: Die acht Bytes ab ad im Floppyspeicher werden als Hexdump dargestellt. Änderungen können mit den Cursortasten und durch Überschreiben vorgenommen werden. Falls ad nicht angegeben ist, wird der letzte M, D oder C-Befehl weitergeführt. Wenn Sie nach dem M-Befehl <RETURN> drücken, können Sie gleich weiterlisten, da der neue M-Befehl gleich vorgegeben wurde. Dies gilt auch für den D und C-Befehl.

C (ad) CHAR: Die 32 Bytes ab ad werden als ASCii Zeichen ausgegeben. Änderungen können nicht vorgenommen werden. Siehe auch M-Befehl.

D (ad) DISASSEMBLE: Der 6502-Befehl ab ad wird disassembliert. Änderungen können nur bei den Hexbytes vor dem Mnemonic vorgenommen werden, sonst wie der M Befehl.

: ad by (by by by by ...) CHANGE: Die (maximal acht) Bytes werden im Floppyspeicher ab ad abgelegt, danach wird ein M-Befehl ausgeführt, um die ASCii Zeichen zu ergänzen. Diesen Befehl brauchen Sie normalerweise nicht einzugeben, da der M-Befehl die Syntax automatisch vorgibt.

L "name" LOAD: Das File name (Typ egal) wird gelistet. Dabei können Sie wählen, ob Sie das File als Basicprogramm listen wollen, als Hexdump wie bei M oder wie bei C als Text. Danach ist (nicht bei Basic) zu entscheiden, ob das File in den ersten beiden Bytes die Startadresse enthält (was gewöhnlich der Fall ist).

G ad GOTO: Das Maschinenprogramm, das im Floppyspeicher ab ad liegt, wird ausgeführt. Es muß mit RTS enden, sonst stürzt das Laufwerk ab (= M-E)

S ad1 ad2 by (by by by ...) SEARCH: Es wird im Floppyspeicher von ad1 bis ad2 einschl. nach den (maximal 28) Bytes gesucht. Sie müssen hintereinander im Speicher stehen. Die Fundstellen werden ausgegeben. Das Suchen geht recht schnell, da es von einem Maschinenprogramm im Floppyram erledigt wird. Es können maximal 254 Fundstellen verwaltetet werden.

Y ad1 ad2 ad3 COPY: Der Bereich von ad1 bis ad2 einschl. im Floppyspeicher wird ab ad3 kopiert (in den Floppyspeicher), ohne etwas zu verändern. So lassen sich Datentabellen kopieren.

T ad1 ad2 ad3 TRANSFER: ad1 - 3 wie bei COPY, zum Kopieren von Maschinenprogrammen, da absolute Adressen, die in den zu kopierenden Bereich zeigen, umgerechnet werden. Für ad2 muß unbedingt das letzte Byte des letzten 6502-Befehles angegeben werden; falls das Programm mit RTS endet, muß also z.B. die Adresse angegeben werden, in der das RTS steht; falls der letzte Befehl JMP ist, muß die Adresse gewählt werden, in der das High-Byte der Sprungadresse steht. Dies ist unbedingt zu beachten, da das Ende sonst nicht erkannt wird.

J JOB LIST: Die Jobs der fünf Puffer werden aufgelistet, dabei wird gleich die Syntax des folgenden J-Befehles gewählt, damit Änderungen einfach durch Überschreiben vorgenommen werden können.

J puf by tr se SET JOB: Für den Puffer puf wird der Jobcode by für Track tr Sektor se festgelegt. Bsp. Lesen von Track 18, Sektor 1 in Puffer 3: J 03 80 12 01 (80 = Code Lesen)

Fehlermeldungen:

FILE NOT FOUND Das File für den L Befehl existiert nicht auf der eingelegten Disk

STACK OVERFLOW Diese Meldung deutet auf einen Programmierfehler im DMS hin. Sie ist aber in langen Tests noch nie aufgetreten und

	würde darauf hinweisen, daß ein interner Stack übergelaufen ist.
STACK UNDERFLOW	Wie STACK OVERFLOW (Stapelunterlauf)
TOO OFTEN FOUND	Beim S-Befehl wurde das Byte so oft (mehr als 254 mal) gefunden, daß ein interner Speicher überlief
DEVICE MISSING	Beim N-Befehl ist das neue Drive nicht bereit oder vorhanden. Die alte Nummer wird wieder übernommen.

Sonstige Meldungen:

ASK ? FOR A HELP	Weist am Anfang auf den ?-Befehl hin
WRITE TO $xxxx	Statusmeldung für die Befehle T und Y
..X FOUND	Anzahl der Fundstellen bei S (dezimal)
N.C.	Puffer nicht benutzt (bei ^)
ERROR #..	Fehler im Puffer (bei ^)
OK	Kein Fehler (bei ^)
>	Prompt (»bereit«)

Hinweise zur praktischen Anwendung finden Sie an anderer Stelle. Hier nur noch die übliche Warnung: Da es sich beim DMS um ein sehr mächtiges Programm handelt, sollten Sie es zum Einarbeiten nur mit Disketten anwenden, auf denen sich keine wichtigen Daten befinden. Durch Eingabe eines falschen Befehles kann im ungünstigen Fall eine gesamte Diskette unbrauchbar und nicht wiederherstellbar werden. Bei den Jobbefehlen befinden Sie sich auf einer Ebene, in der insbesondere die Tracknummer nicht mehr überprüft wird (legale Tracknummern: 1 bis 35, in einigen Fällen bis 41). Versuchen Sie also, etwa von Track 150 zu lesen, so wird die Floppy diesen Befehl brav ausführen und versuchen, den Lesekopf auf die Spur 150 zu positionieren. Da nach Spur 41 ein mechanischer Anschlag kommt, sind dann ernsthafte Störungen wie verstellte oder gar zerstörte Lesekopf, Lesemechanik und/oder Schrittmotor zu befürchten. Verfasser und IPV Verlag AG übernehmen keine Gewähr für zerstörte Disketten oder Hardware. Gerne stehen wir jedoch für Ausfünfte zur Verfügung.

Nun für den Tüftler noch die hexadezimale Speicherbelegung:

1. Computer

00FB-00FC	Adresszwischenspeicher
61A8-61AA	Sprung zur Startroutine bei $689A
61AB-62AA	Tabelle: Befehlscode des aktuellen Bytes
62AB-63AA	Tabelle: Adressierungsarten
63AB-63B7	Tabelle: Anz. Bytes für jede Adr.art
63B8-63D1	Adressen der Routinen zur Anzeige der Adr.arten
63D2-649A	Texte der Mnemonics
649B-650A	Startmeldung
650B-650C	»UJ« für »R« Befehl
650D-6513	Text für den reversen Prompt bei Fehlern
6514-6548	STACK OVERFLOW, STACK UNDERFLOW, TOO OFTEN FOUND
6549-6589	Texte für »J« und »^« Befehle
658A-658B	»I0« zum Initialisiern der Floppy
658C-659D	Texte für »S« Befehl
659E-65AD	»0123456789ABCDEF« für Hexzahlen
65AE-65B9	»WRITE TO $« für T und Y Befehle
65BA-65BE	Lesetext (UC 00 00 00)
65BF-65C6	Texte für die Vorgabe der Defaultwerte
65C7-65CC	»M-W« 00 00 00
65CD-65F9	Namen der Jobs für »^« Befehl
65FA-6607	Tabelle mit den Zeigern auf diese Namen
6608-660C	»M-R« 00 00

660D-661C	»OK«, »N.C.« und »ERROR #« für »^« Befehl
661D-6621	»M-R« Befehl zum Test auf Masch.prog.
6622-6626	»M-E« 00 00 für »G« Befehl
6627-664E	Text für »L« Befehl
664F-666E	»DEVICE MISSING«, »FILE NOT FOUND«
666F-668F	Text 2 für »L« Befehl
6690-6696	»M-W« 00 00 01 00
6697	Devicenummer (8) für »N« Befehl oder SYS 25000,X
6698	Flag für Anführungszeichen für INPUT Routine
6699-669A	Zwischenspeicher für Adresse beim »D« Befehl
669B	Zwischenspeicher
669C	Stackpointer für internen Stack ab $669E
669D	Zeiger auf Eingabezeile
669E-66A7	Stack, auf dem das X Register abgelegt wird
66A8-66AF	Latch, etwa für »M«, »D«, »Y« oder »T« Befehle
66B0	Zwischenspeicher
66B1	Name des aktuellen Befehles
66B2-66C7	Tabelle mit den Namen der Befehle
66C8-66F1	Adressen der Routinen der Befehle
66F2-672B	Maschinenprogramm für Floppypuffer 2 zum Lesen von Werten aus dem Floppyspeicher (siehe Einleitung)
672C-678B	Maschinenprogramm für Floppypuffer 2 zum Suchen von Werten im Floppyspeicher (»S«-Befehl)
678C-6899	Hilfstexte für »?« Befehl
689A-68F7	Maschinenprogramm: Hauptprogramm
68F1-68F3	am Anfang JSR 0, Sprung in die Befehlsroutinen
68F8-6B69	Unterprogramme (Ein- Ausgabe, Umrechnungen etc.)
6B6A-7553	Routinen der Befehle
7554-7653	Eingabepuffer

2. Floppyspeicher

0014-0015	zu lesende Adresse beim »S« Befehl
006F-0070	zu lesende Adresse
0300-03FF	LOW Bytes der Funstellen
04FE	Flag, ob TOO OFTEN FOUND (dann 255)
04FF	Anzahl Funstellen
0500-052D	Maschinenprogramm 1 als verbesserter »M-R« Befehl
0500-054D	Maschinenprogramm 2 für »S« Befehl
0600-0699	High-Bytes der Funstellen

Tester 1541 - Prüfprogramm für Rotationsgeschwindigkeit und Kopfjustage

Das Disketten-Laufwerk ist mit Sicherheit Ihr wichtigstes Peripheriegerät. Da es mechanische Präzisionsteile enthält, ist es auch das, was am ehesten Kummer bereitet. Der »Tester 1541« hilft Ihnen dabei, diese Probleme frühzeitig zu erkennen und zu lokalisieren.

1. Warum?

Für viele Anwender ist das Laufwerk die wichtigste Komponente des Gesamt-Systems - es ist das am häufigsten zur Programmspeicherung genutzte Medium. Ohne die Floppy wären die meisten Computer unpraktisch, wenn nicht unmöglich.

Wenn Ihr Laufwerk 1541 nicht mehr so arbeitet, wie Sie es von ihm erwarten, kann Ihnen der »Tester 1541« dabei helfen, herauszufinden, wo der Fehler steckt. Das Programm analysiert die kritischten Funktionen der Drive-Mechanik und versorgt Sie am Bildschirm mit wichtigen Informationen. Auf einen Blick sehen Sie, wie schnell sich die Diskette dreht, wie verstellt der Schreib-/Lesekopf ist, und wie gut er jeden Track erreicht. Es erscheint auf Wunsch sogar eine Angabe darüber, wie es mit der Lesbarkeit der Bereiche zwischen den Tracks steht (sog. »Halftracks«). Wenn diese Informationen erst einmal am Bildschirm stehen, sollte es keine Probleme mehr geben, zu entscheiden, ob das Problem gravierend ist und ob eine Reparatur notwendig ist. Wenn Sie schon wissen, wo der Fehler steckt, sparen Sie eine Menge Geld und Zeit.

Das Laufwerk 1541 ist ein ziemlich zuverlässiges Gerät. Seine Mechanik ist stabil und hält meistens auch gröberen Umwelteinflüssen tapfer stand. Allerdings ist kein Ding auf dieser Welt perfekt, und so gibt es immer irgendwann einmal den Augenblick, in dem etwas in der 1541 »in die ewigen Jagdgründe eingeht« oder schlicht und einfach nur »spinnt«. Typische Probleme ergeben sich durch defekte Disketten, defekte oder verstellte (dejustierte) Mechanik, oder Fehler in der Elektronik. All diese bewirken dann Störungen beim Laden oder Speichern von Dateien, in Extremfällen funktioniert das Laufwerk gar nicht mehr. In diesem Fall sollten Sie es aus- und wieder einschalten. Prüfen Sie als nächstes alle Steckerverbindungen, die Stromzuleitung sowohl an der Steckdose wie auch an der Laufwerks-Rückseite, um sich zu vergewissern, daß alles fest verbunden ist. Bringt das nichts, prüfen Sie die hinten befindliche Sicherung. Falls notwendig, ersetzen Sie die Sicherung durch eine vom genau gleichen Typ.

Wenn keiner dieser Schritte das Problem löst, stimmt wahrscheinlich etwas mit der eingebauten Elektronik nicht. Sollte nach dem Einschalten die rote Leuchtdiode nicht ausgehen oder zu blinken beginnen, stimmt mit ziemlicher Sicherheit etwas mit der Elektronik nicht. In solchen Fällen ist der Gang zur Reparaturwerkstatt leider unumgänglich. Andere Probleme aufzuspüren ist normalerweise nicht ganz so einfach. In den meisten Fällen wirken sich Störungen in Form der bekannten Read-Errors aus, die rote Lampe flimmert und das Laufwerk erzeugt ein Rattern. Das Laden oder Speichern eines Programmes dauert dann länger als gewöhnlich, oder die 1541 bricht gar die ganze Prozedur ab, die rote LED blinkt um einen Fehler zu melden. Sie können natürlich leicht herausfinden, um welchen Fehler es sich handelt, indem Sie das folgende Programm starten:

```
10 OPEN15,8,15:INPUT#15,A,B$,C,D:PRINTA;B$;C;D:CLOSE15:END
```

Manchmal ist es nützlich, die Fehlermeldung zu kennen, aber im allgemeinen reicht das nicht, um die wirkliche Fehlerquelle herauszufinden. Wir haben es mit dieser Fehlergattung nicht nur bei defekten Disketten zu tun, sondern auch wenn die Rotationsgeschwindigkeit der Magnetscheibe nicht stimmt, wenn der Lesekopf dejustiert ist, oder sogar wenn Sie das Laufwerk im Störfeld eines Fernsehers, Monitors, Computers oder ähnlichem betreiben. Da diese Pannen nahezu die gleichen Symptome zeigen, brauchen Sie ein Programm wie den »Tester 1541«, um weitere Informationen über die genaue Art und Ursache des Fehlers zu gewinnen.

2. Das Programm

Das Programm ist aus verschiedenen Gründen vollständig in Assembler geschrieben. Dennoch brauchen Sie keinerlei Maschinensprache-Kenntnisse, um damit umgehen zu können. Das Programm läßt sich wie ein Basicprogramm laden, starten und ggf. kopieren. Nach der Eingabe sollten Sie eine Kopie auf Diskette speichern. Besitzen Sie eine Datasette? Dann speichern Sie doch auch eine Kopie auf Band, damit das Utility im Falle eines Falles geladen werden kann, auch ohne das Diskettenlaufwerk. Um den Tester von Diskette zu laden und starten, geben Sie ein:

```
LOAD "TESTER 1541",8
RUN
```

Das Titelbild erscheint, nach einer kurzen Wartezeit von knapp einer Sekunde folgt das Hauptmenü. In der Pause überträgt der Tester ein Maschinenprogramm in den Floppy-Speicher. Jetzt haben Sie die Wahl zwischen dem Rotations-Geschwindigkeits-Test oder dem Justage-Test. Mit <Q> wird das gesamte Programm beendet, die Floppy wird initialisiert, nach einigen Sekunden sind Sie wieder im Direktmodus. Der Tester läßt sich ggf. mit RUN oder SYS 2061 wieder starten.

3. Rotationsgeschwindigkeit

Der Speed-Test prüft, wie schnell die Magenetscheibe vom Hauptmotor des Laufwerks um sich selbst gedreht wird, wenn die 1541 liest oder schreibt. Die Geschwindigkeit darf leichte Toleranzen aufweisen, ohne daß sich Probleme ergeben. Es gibt da allerdings Grenzen, und einige Programme - vor allem mit Kopierschutz - sind schon arg »pingelig«, wenn es um den Diskettenzugriff geht. Gewöhnlich sollte die Diskette sich mit 300 rpm (Umdrehungen pro Minute, Rotations per Minute) drehen, das entspricht einer Umlaufzeit von 0,2 Sekunden oder einer Frequenz von 5 Hertz (Winkelgeschwindigkeit 31,4 Hz, Bahngeschwindigkeit 2,094 Meter pro Sekunde). Dieser Wert sollte im Normalfall um nicht mehr als 1 oder 2 rpm nach oben oder unten abweichen. Falls doch, treten Lesefehler auf.

Um die Rotationsgeschwindigkeit zu messen, laden und starten Sie das Programm. Im Hauptmenü gelangt man mit der Taste <1> in den entsprechenden Programmpunkt. Sie sollen dann eine leere Diskette einlegen. Diese kann formatiert oder unformatiert sein, wichtig ist aber, daß sie nicht schreibgeschützt ist. Das Programm beschreibt die Diskette, allerdings auf einem Bereich, der sonst nicht für die Datenspeicherung verwendet wird (ab Track 36). Sollten sich Daten auf der Floppy befinden, werden diese nicht zerstört. Um ganz sicherzugehen, benutzen Sie eine Diskette, auf der sich nichts Wichtiges befindet. Mit <Q> können Sie abbrechen.

Nach dem Einlegen starten Sie mit einer beliebigen Taste den Meßvorgang. Die Bildschirmmaske für den Geschwindigkeits-Test erscheint, und das Drive läuft an. Nach einigen Berechnungen zeigt der Tester die Rotationsgeschwindigkeit in rpm mit dem Sollwert 300.0 und der errechneten Differenz gegen 300.0 rpm an. Diese enthält auch ein Vorzeichen (+ oder -), je nachdem, ob der gemessene Wert nach oben oder unten abweicht. Ein Plus zeigt eine zu hohe Geschwindigkeit an, bei Minus dreht sich die Floppy zu langsam. Ist der Wert in den erlaubten Grenzen (Abweichung +- 2 rpm), wird er in weiß geschrieben. Liegt er im »kritischen« Bereich zwischen 2 und 4 rpm, erscheint er in gelb. Abweichungen über 4 rpm bedeuten grobe Fehlfunktion und werden in rot dargestellt. In diesem Fall kann das Laufwerk zwar vielleicht noch laden und speichern, hat aber wahrscheinlich Probleme, Diskette zu bearbeiten, die auf

anderen Laufwerken formatiert wurden. Aus diesem Grunde sollten Sie das Laufwerk justieren lassen, wenn die Geschwindigkeit ständig in rot angezeigt wird.

Dieser Wert wird ständig neu gemessen und in der Tabelle angezeigt. Ist die unterste Zeile erreicht, scrollt die Tabelle eine Zeile nach oben. Es kommt vor, daß die Geschwindigkeit leicht schwankt, während die 1541 aktiv ist. Es gibt dann keinen Grund zur Unruhe - manche Disketten sind besser als andere, und auch Disketten selbst produzieren Schwankungen, je nachdem, wieviel Reibung sie erzeugen. Stimmt die Drive-Geschwindigkeit im Normalfall, bei bestimmten Disketten jedoch nicht, sollten Sie diese speziellen Disketten nicht mehr benutzen. Die Rotationsgeschwindigkeit verändert sich auch, wenn der Keilriemen in der 1541 schlüpft. Das kommt durchaus vor, vor allem auf alten oder stark genutzten Geräten. In diesem Fall sollte der Riemen ausgewechselt werden.

Beim Speed-Test können auch Fehler auftreten. Diese werden auf dem Bildschirm mit einer Fehlernummer gemeldet, die genau der der normalen 1541-Fehlermeldungen entspricht. Die Nummer 26 bedeutet etwa, daß die Diskette schreibgeschützt ist, Nummer 21 weist auf eine fehlende Diskette oder einen zerstörten Kopf hin. Oder haben Sie vergessen, die Laufwerksklappe zu schließen? Tritt ein Fehler auf, sollen Sie eine Taste drücken.

Um den Test zu stoppen, ist ebenfalls ein Tastendruck notwendig. Unter Umständen müssen Sie die Taste mehr als einmal oder etwas länger drücken, da der Tester intensiv mit der 1541 kommuniziert und daher die Tastaturabfrage nur unregelmäßig durchführt. Am besten drücken Sie die Leertaste, diese hat eine automatische Tastenwiederholung.

4. Kopfjustage-Test

Um die Kopfjustage zu prüfen, drücken Sie im Hauptmenü die Zifferntaste <2>. Jetzt soll die Testdiskette eingelegt werden (diesmal muß sie wirklich formatiert sein, sonst schlägt der Justage-Test fehl). Dann werden wieder einige Wahlmöglichkeiten aufgelistet. Die für diesen Test verwendete Diskette kann irgendeine formatierte Diskette sein. Im Interesse eines sorgfältigen Tests ist es wichtig, daß diese Diskette auf einem absolut perfekt justierten Laufwerk formatiert wurde. Je »besser« formatiert die Testdiskette ist, desto genauer wird der Test. Eine kommerziell produzierte Diskette (beispielsweise eine Leserservicediskette oder die Diskette aus einem neueren 64'er Sonderheft) ist ideal. Benutzen Sie keine kopiergeschützte Floppy. Zwar schreibt der Justage-Test nicht auf die Diskette, so daß sicher kein darauf gespeichertes Programm angegriffen wird. Dennoch sollten Sie Kopien von wichtigen Files machen.

Nachdem die Testdisk eingelegt wurde, wird der Test mit der <J>-Taste gestartet. Sie sehen einen Datenschirm mit zwei Bereichen zu je vier Spalten. Die Spalten sind überschrieben mit A) geprüfter Track, B) gelesener Track, C) Track-Lesbarkeit und - falls eingeschaltet - D) Halftrack-Lesbarkeit. Wenn der Test läuft, werden die 35 Tracks der Diskette in diese Tabelle eingetragen. Nach Track 18 wird der linke Teil rechts fortgesetzt.

Über der ersten Spalte steht die Kennung für »geprüfter Track«. Hier zeigt der Tester die Nummer des momentan geprüften Tracks an. Es beginnt immer mit Track 1 und endet mit Track 35. Dies ist das Standard-Format der 1541 und ändert sich nicht, gleichgültig, was auf der Diskette gespeichert ist und ob

das Laufwerk justiert ist oder nicht.

Bevor die Daten für Track 1 erscheinen, durchläuft das Programm eine komplizierte Prozedur, um festzustellen, auf welchem Track sich der Kopf momentan befindet. Danach wird der Kopf heruntergefahren, auf Track 1 und gegen die Kopfbegrenzung. Verlief die Ermittlung der Position vorher einwandfrei, sollte es ein leises »Klicken« geben, wenn der Kopf die Sperre erreicht. Konnte die Position nicht ermittelt werden, wird ein »Bump« ausgeführt, um den Kopf ganz nach unten zu ziehen. Das bekannte Rattern ist zu hören. Das ist hier nur sehr selten der Fall, kann aber vorkommen, wenn die 1541 total dejustiert ist oder Sie versehentlich eine unformatierte Diskette verwenden.

Über der zweiten Spalte steht »gelesener Track«. Hier finden Sie die Nummer des Tracks, auf dem der Lesekopf im Augenblick steht. Auf jedem Track einer formatierten Diskette findet sich auch eine Angabe über die Tracknummer. Das Testprogramm versucht, diese Angabe zu lesen und zeigt sie in Spalte 2 an. Die Zahlen in den Spalten 1 und 2 sollten für jeden Track identisch sein. In diesem Fall werden sie in weiß ausgegeben. Das bedeutet, daß der richtige Track gelesen wurde. Stimmt die von Diskette gelesene Tracknummer nicht mit Spalte 1 überein, wird die Zahl in rot ausgegeben. Im Normalfall sind entweder alle Zahlen in Spalte 2 weiß oder alle rot. Es ist unwahrscheinlich, daß sowohl rote wie weiße Werte vermischt zu sehen sind.

Ein falsch justierter Lesekopf ist der Übeltäter, wenn in der zweiten Spalte rote Zahlen erscheinen. Die Differenz zeigt Ihnen, wie stark der Kopf dejustiert ist. Ist die Differenz 1, so beträgt die Dejustage einen Track. Zwei Tracks sind es bei einer Differenz von 2, und so weiter. Auch eine Aussage über die Richtung der Dejustage ist möglich. Wenn die Zahlen in der zweiten Spalte größer sind als die ganz links, befindet sich der Kopfstop zu weit oben und der Kopf kann die Tracks mit niedrigen Nummern nicht mehr anfahren. Sie wären dann nicht imstande, ein Programm, das auf Track 1 gespeichert ist, zu laden. Die weiter verbreitete Störung ist, daß die Zahlen in der zweiten Spalte kleiner sind als die in der erste Spalte. Der Kopf kann sich dann zu weit nach unten bewegen. Das stellt kein Problem dar, wenn die Disketten ordnungsgemäß formatiert wurden, da das Laufwerk den Kopf auf alle Tracks positionieren kann. Auch wenn Disketten auf einem Laufwerk mit einem Justierungsproblem der Kopfsperre formatiert wurden, sollte es kein Problem beim Lesen aller Tracks geben, solange die Floppy mit dem selben Gerät bearbeitet wird. Wenn sie dann allerdings in ein Laufwerk mit korrekt eingestellter Kopfsperre eingelegt wird, können die Tracks nicht mehr gelesen werden, die zu weit unten formatiert wurden.

Ohne einen Justagetester ist es sehr schwer, einem dejustierten Kopf auf die Schliche zu kommen. Er macht sich nur bemerkbar, wenn Disketten auf mehr als einem Laufwerk benutzt werden, oder wenn sich die Diskette mit Dateien füllt. Wenn die Diskette nicht allzu voll ist, sind keine Störungen zu erwarten. Die Programme werden auf der Scheibe beginnend auf den mitteleren Tracks abgelegt und arbeiten sich dann nach außen in Richtung Track 1 und Track 35. Sogar wenn das Laufwerk einen Fehler erkent und den Kopf gegen die Begrenzung fährt, um den Fehler zu korrigieren, so lange der Kopf sich selbst immer wieder korrigieren kann, ist alles in Ordnung. Es ist schön, daß die 1541 in der Lage ist, solche Korrekturen selbständig vorzunehmen. Aber dadurch wird die Erkennung eines verstellten Kopfes erschwert, es sei denn, Sie arbeiten mit einem Spezialprogramm wie diesem. Wenn das Problem erst einmal erkannt ist, sollten Sie den Kopf neu justieren lassen, um den Schaden möglichst gering zu halten.

Über der dritten Spalte steht »Track-Lesbarkeit«. Die Daten, die hier angezeigt werden, sind am wichtigsten bei der Beurteilung, ob das Laufwerk ein Justierungsproblem hat oder nicht. Wie aus der Überschrift bereits hervorgeht, gibt das Programm hier an, wie gut das Drive in der Lage ist, einen Track zu lesen. Das Testprogramm versucht, mindestens 17 Sektoren auf jedem Track zu lesen. Gelang es bei jedem Sektor beim ersten Versuch, den Datenheader zu lesen, beträgt die Lesbarkeit 100%. Dieser Wert wird weiß angezeigt. So sollte die Angabe für jeden Track aussehen, wenn das Laufwerk richtig eingestellt ist. Ist mehr als ein Versuch notwendig, um einen der Header zu lesen, wird die Angabe in der dritten Spalte eine Zahl unter 100 sein. Liegt sie im Bereich zwischen 94 und 99 %, wird sie gelb angezeigt. Das bedeutet, daß alle Header im Prinzip gelesen werden konnten, aber einige Schwierigkeiten auftraten. Ist die Lesbarkeit unter 94 %, wird sie rot ausgegeben. Ein klares Zeichen dafür, daß es Probleme beim Lesen eines oder mehrerer Header gab, oder das Laufwerk konnte einige Sektoren überhaupt nicht lesen.

Ein justiertes Laufwerk sollte keinerlei Probleme haben, auf allen Tracks den Wert 100 % zu erreichen. Ist dies nicht der Fall, probieren Sie es mit einer anderen Testdiskette. Treten hier ähnliche Ergebnisse auf, können Sie davon ausgehen, daß Ihr Laufwerk ein Justierungs-Problem hat. Bevor Sie es allerdings zur Justage weggeben, sollten Sie es zunächst einmal an einem anderen Ort betreiben, möglichst weit weg von Ihrem Fernseher, Monitor, Drucker, Computr und anderen elektronischen Geräten, die sich vielleicht in der näheren Umgebung befinden. Elektronische Interferenzen sind wahrscheinlicher als echte Dejustierung, und zeigen genau die gleichen Symptome. Sollte die »Versetzung« keinen Erfolg bringen, bleibt Ihnen der Weg in die Reparaturwerkstatt zwecks Justage nicht erspart.

Die vierte Spalte trägt die Bezeichnung »Halftrack Lesbarkeit«. Es handelt sich um ähnliche Informationen wie die in der dritten Spalte, allerdings wurde der Kopf hier zwischen zwei Tracks gestellt. Der Schrittmotor, der im Laufwerk den Kopf antreibt, muß zwei Schritte machen, um einen vollen Track zu überspringen. Macht er nur einen Schritt, befindet sich der Tonkopf zwischen zwei Spuren. Im Idealfall sollte die 1541 nicht in der Lage sein, Daten zwischen zwei Tracks zu finden, die Lesbarkeit sollte also immer gleich Null sein. In der Praxis »strahlen« einige der Daten der benachbarten Tracks in den Bereich zwischen zwei Spuren. Auch wenn das Gerät imstande ist, Daten in den »Halftracks« zu lesen, wäre es verständlich, wenn es die größten Schwierigkeiten hätte, wenn der Kopf sich exakt zwischen zwei Spuren befinden und die Lesbarkeit zu den benachbarten Tracks hin zunimmt. In Wirklichkeit stimmt das so weit, daß man nützliche Informationen daraus ableiten kann, aber wegen Toleranzen in der Mechanik und in der Magnetscheibe selbst können diese Werte nicht als absolut angesehen werden und sollten nur zum Vergleich herangezogen werden. Da außerdem aufgrund der vielen notwendigen Leseversuche die Ermittlung der Halftrack-Werte ziemlich lang (bis zu einer oder zwei Sekunden pro Track) dauern kann, besteht die Möglichkeit, die Ausgabe der vierten Spalte ganz abzuschalten. Dazu betätigen Sie im Untermenü »Justage« die <H>-Taste, um die Halftrack-Option abzuschalten.

Die Farbgebung in der vierten Spalte ist anders als die in Spalte 3. Ist die Lesbarkeit 0 %, wird dieser Wert weiß ausgegeben. Dies ist das Ideal beim Lesen von Halftracks. Werte zwischen 1 und 80 % färbt das Programm gelb, die Werte fallen gewöhnlich in diesen Bereich. Rote Zahlen (81 bis 100 %) sind seltener als gelbe, erscheinen aber auch auf Stationen, die sonst sehr gut arbeiten. Die Justierung ist am besten, wenn wenigstens einige Nuller in der vierten Spalte auftauchen. Sind viele Werte rot, könnten Probleme vorhanden

sein, seien Sie aber nicht besorgt, wenn alle Zahlen in der dritten Spalte weiß sind.

Es wird Ihnen auch auffallen, daß hinter der vierten Spalten in vielen Fällen ein Plus- oder Minuszeichen steht. Das Testprogramm zählt, wie oft es Daten vom unter dem Halftrack liegenden Track liest, und wie oft Daten von der oberen Nachbarspur. Es zeigt dann ein »-« oder »+« an, um anzuzeigen, welche Spur öfter gelesen wurde, und setzt Sie in Kenntnis, welcher Nachbartrack näher bei dem vermeintlichen Halftrack lag. Wird ein Minuszeichen angezeigt, erscheint der niedrigere Track näher. Wird das Plus gemeldet, wurde öfter vom höheren Track gelesen. Ist das Vorzeichen für alle 35 Tracks das gleiche, scheint der Lesekopf sich stark an der oberen oder unteren Trackgrenze zu befinden, je nachdem, welches Zeichen erscheint. Ist kein Symbol zu sehen, erschienen entweder beide Tracks gleich weit entfernt, oder es konnten zu wenige Werte für den Vergleich gelesen werden.

Obwohl der Justagetest eine beträchtliche Menge an Informationen liefert, ist er nicht schwer in der Anwendung, insbesondere, wenn Sie einen Farbmonitor benutzen. Sind alle Zahlen in den Spalten 2 und 3 weiß, ist die Justage völlig in Ordnung. Bei der Farbgebung im Programm wurde aber bewußt darauf geachtet, daß auch auf Monochrom-Monitoren (wie ihn der Programmautor benutzt) klare Unterschiede zu erkennen sind. Die weißen Zahlen sind am hellsten, die gelben etwas dunkler und die roten heben sich klar von den übrigen ab. Die Benutzerführung erfolgt übrigens in dunkelgrau. Die Spalten 1 und 2 sollten identisch sein, und Spalte 3 sollte vollständig mit einer 100 gefüllt sein, dann stimmt alles. Eine Reparatur wäre anzuraten, wenn die Spalten 1 und 2 nicht übereinstimmen oder in Spalte 3 Werte unter 94 % stehen. Bevor Sie das Laufwerk weggeben, probieren Sie aber unbedingt die Prozedur an mehreren Testdisketten durch, um sicherzugehen, daß nicht einfach nur die Diskette defekt war.

Die Justagetest-Funktion läßt sich auf Tastendruck abbrechen. Das Programm kehrt dann in das Justage-Menü zurück, von wo Sie mit <Q> wieder das Hauptmenü erreichen. Unter Umständen muß auch hier die Taste mehrmals bzw. längere Zeit betätigt werden, da der Computer mit dem Laufwerk kommuniziert und von Zeit zu Zeit Tastendrücke ignoriert.

Das Programm bricht den Test von sich aus ab, wenn es Track 35 erreicht hat. Drücken Sie dann eine Taste, um das Menü wieder zu erreichen. Sie können den Test auch in einer Endlosschleife durchführen, indem Sie ihn im Justage-Menü nicht mit <J>, sondern mit <W> starten. Mit dieser Option löscht das Programm nach Track 35 automatisch den Bildschirm und fährt wieder mit Track 1 fort. Hier müssen Sie (möglicherweise mehrmals) eine Taste drücken, um abzubrechen.

Bleibt noch die Taste <1> im Justage-Menü. Diese Funktion bewegt den Kopf einfach nur auf Track 1 (bzw. dort, wo dieser sein sollte) und parkt ihn dort, bis Sie eine Taste drücken. Diese Möglichkeit wurde für diejenigen eingebaut, die dieses Programm dazu benutzen wollen, den Kopf selbst zu justieren. Diese Justage erfordert allerdings ein Ausbauen des Laufwerks und sehr präzise Arbeiten an der Drive-Mechanik. Da außerdem die Gefahr eines elektrischen Schlages besteht (grundsätzlich stehen ALLE Metallteile im Innern der 1541 unter absolut tödlicher Hochspannung!), sollte diese Arbeit nur ein Fachmann ausführen.

Weiter besteht im Justage-Menü, wie erwähnt, die Möglichkeit, die Halftrack-Prüfung abzuschalten. Die Taste <H> wirkt hier wie ein Schalter. Je nachdem, ob das Wort »AN« oder »AUS« unterstrichen ist, werden die

Zwischen-Tracks getestet oder nicht.

5. Nachspiel

Um das gesamte Programm zu verlassen, sollten Sie in jedem Fall vorher ins Hauptmenü zurückkehren (von den beiden Untermenüs jeweils mit der <Q>-Taste) und dort ebenfalls mit <Q> das Testprogramm beenden. Dadurch wird das Drive initialisiert und ist danach wieder in Bereitschaft für das nächste Programm. Sie dürfen das Laufwerk nach dem Start des Testprogrammes übrigens nicht abschalten, da sonst das in dessen RAM enthaltene Maschinenprogramm verloren geht. In diesem Fall muß der Tester vor der weiteren Arbeit erst wieder gestartet werden.

Sie können die Justierung des Laufwerks eines Freundes testen, ohne dazu das Laufwerk selbst zu benötigen. Lassen Sie den Freund eine Diskette auf seinem Drive formatieren, und verwenden Sie diese dann als Testdisk für den Justage-Test. Ergibt der Test ein positives Ergebnis, ist die Justierung des Drives wahrscheinlich in Ordnung. Verläuft der Test negativ, sollten Sie das Programm direkt auf dem fremden Laufwerk ausprobieren, bevor eine Entscheidung über die Reparatur getroffen wird.

Jede Diskette läßt sich auf korrekte Formatierung überprüfen, indem sie einfach als Testdisk für den Justage-Test verwendet wird. Die Daten, die das Programm ausgibt, zeigen Ihnen, ob alle Tracks formatiert wurden, ob sie in der richtigen Reihenfolge sind, und ob bestimmte Spuren Fehler aufweisen (dann nämlich, wenn die Lesbarkeit einen kleineren Wert hat als sie es sollte).

Es gibt noch viele weitere Anwendungsmöglichkeiten, als die, die wir hier aufgezählt haben. Das Programm eignet sich vielleicht nicht für jede Anwendung, die Ihnen jetzt so vorschwebt, aber es enthält eine gut entwickelte »Alarmanlage«, die Ihnen ganz wesentlich dabei hilft, das Laufwerk vor Fehlern zu schützen. Es ist nicht möglich, das Laufwerk durch falsche Programmierung zu zerstören, aber es ist möglich, es zum Absturz zu bringen. In diesem Fall schalten Sie es kurz aus und wieder ein, und geben folgenden Befehl:

```
OPEN 1,8,15,"I0":CLOSE 1
```

Dadurch wird der Schreib-/Lesekopf auf seine Standard-Position gebracht und das Laufwerk initialisiert.

6. So funktioniert das Testprogramm

Leider können wir hier aus Platzgründen keine detaillierte Beschreibung der genauen internen Vorgänge liefern, wohl aber eine grobe Übersicht. Der wichtigste Bestandteil ist das Maschinenprogramm, das beim Start in das Floppy-RAM übertragen wird. Dieses wird vom im C 64 befindlichen Maschinenprogramm aufgerufen und führt unter anderem die Messungen durch, dient aber auch dazu, die Motoren zu steuern. Der Aufruf erfolgt mittels M-E Befehl über eine Sprungtabelle, die in der Floppy ab $400 steht und folgende Befehle bietet:

```
$400  ersten Track testen, später alle anderen
$403  nächsten Halftrack testen
$406  nächsten Fulltrack testen
$409  Kopf von Halftrack auf Track 18 positionieren
$40c  Ende, Kopf auf Track 18
```

$40f Rotat. Geschw. erneut berechnen
$412 Rotat. Geschw. berechnen
$415 nur Track 1 lesen
$418 LED und Hauptmotor aus
$41b Hauptmotor an
$5b3 Kopf auf Fulltrack positionieren

Beim Justage-Test geht das Programm die gewünschten Tracks durch und versucht, 17 Sektoren zu lesen. Die Floppy übergibt dem C 64 eine Tabelle mit den Leseversuchen für die einzelnen Sektoren. Daraus berechnet der C 64 die Prozentzahl. In einer separaten Speicherzelle wird die gelesene Tracknummr übergeben. Diese Daten übernimmt der Computer mit dem M-R Befehl, eigene Busroutine sind nicht vorhanden.

Zum Geschwindigkeitstest legt das Programm auf Track 36 oder höher einen Killertrack an, der nur aus einer langen Sync-Marke und einer ganz speziellen Markierung an einer Stelle besteht. Um die Geschwindigkeit zu messen, wartet das Programm auf diese Markierung und setzt dann einen Zähler auf Null. Dieser wird nun mit jeder Sync-Marke um eins erhöht, so lange, bis die Markierung wieder kommt. Der Zählerstand gibt dann indirekt schon die Rotationsgeschwindigkeit, der genaue Wert in RPM wird unter Berücksichtigung der Floppy-Taktfrequenz und der Länge des zuständigen Maschinenprogrammes aus- und in RPM umgerechnet. Nähere Informationen entnehme der interessierte Leser auch der Speicherbelegungstabelle.

$0002-0005 temporär
$0006   Tracknummer
$00b5   Flag: Halftracks anzeigen
$00b6   Justage-Test Modus
$00f7   Zähler
$00f8   Zehntel
$00f9-00fa Geschwindigkeitswert
$02c0   Übergabespeicher für Diskfehler
$02c1-02c4 Übergabespeicher für Geschwindigkeit
$0334-0347 Übergabespeicher für Justage-Test
$0801-15b9 Programm Tester 1541
$0801-080d Basic-Kopf
$080e   Initialzündung: Sprung nach $0ed9
$0810-0818 Floppybefehle
$0819-0828 Hexadezimalzahlen
$0829-0cd4 Bildschirmtexte
$0cd5-130a C 64-Maschinenprogramm
$0cd5   Rahmen zeichnen
$0d29   diverse Bildschirmroutinen
$0d92   Hexadezimalzahl ausgeben
$0deb   Peripheriegerät ansprechen
$0e06   Disk-Befehl ausführen
$0e33   Maschinenprogramm in Floppy starten
$0e4b   Zahl formatieren
$0e5e   Justage-Ergebnisse abfragen und auswerten
$0ed9   Hauptprogramm
$0f4f   Hauptmenü
$0f71   Beenden
$0fa1   Rotations-Geschwindigkeit auswerten
$0fe1   Hauptschleife dazu
$102b   Geschw. berechnen
$1125   Umrechnung -> RPM
$119a   Justagetest

$11c2   Justage-Menü
$11ed   Justage-Schleife
$121d   nächster Track
$130a   letztes Byte für C 64
$130b-15b9 Floppy-Maschinenprogramm

5. KAPITEL: SONSTIGES

DOC 64 2.2 - ein Selbsttest für den C 64 mit allen Schikanen

Spätestens, wenn er »nicht mehr geht« sorgt man sich um das Innenleben des Computers. Manchmal genügt es, wenn ein Chip ausgetauscht werden muß. Welcher das ist, läßt sich leicht mit Hilfe einer Selbsttest-Routine feststellen, die Komponenten des Rechners auf Funktionsweise prüft. »DOC 64«, der Name leitet sich von »Doctor 64« ab, enthält ungewöhnlich komplexe Testroutinen, die wirklich jede erdenkliche Funktion genauestens prüfen. Neben dem obligatorischen RAM- und ROM-Test (dabei werden sogar verschiedene Kernal-Versionen berücksichtigt und ggf. die Chip-Nummer des defekten RAMs angezeigt) prüft DOC 64 auch den Prozessor 6510, den Soundchip SID, den Videobaustein VIC, die beiden »Ein-Ausgabeagenten« (CIAs) und mit deren Hilfe das gesamte System-Timing sowie unter Zuhilfenahme von sog. »Dongles« einige Schnittstellen. Ingesamt muß der Brotkasten über 30 Tests durchlaufen, bevor er die Funktionsfähigkeit bescheinigt bekommt. Natürlich werden die Einzelergebnisse angezeigt.

Ein solches Programm ist nicht nur für den Privatmann interessant, sei es im Falle eines Defektes oder auch nur, um regelmäßig den Rechner durchzutesten. Vorstellbar wäre auch die Anwendung in einer Reparatur-Werkstatt, um Defekte leichter zu finden, oder beim Kauf eines neuen C 64, wenn ermittelt werden soll, ob das neue Gerät das tun wird, was man von ihm erwartet.

Bevor wir die einzelnen Programmfunktionen beschreiben, zunächst noch einige Anmerkungen zu den »Dongles«. Damit der C 64 die Schnittstellen testen kann, müssen vor dem Test einige Stecker an den Userport, Cassettenport und die beiden Joystickports (Control-Ports) gesteckt werden. Diese enthalten nur einige Verbinden bzw. wenige passive Bauelemente, die zur Prüfung der Schnittstellen benötigt werden. Die drei Dongles können Sie selbst nach dem Schaltplan nachbauen. Für den seriellen Port wird kein Dongle benötigt, da hier im Computer bereits die Voraussetzungen erfüllt sind. Sie können DOC 64 aber auch völlig ohne Zusatz-Hardware betreiben, allerdings werden dann die Schnittstellen-Test negativ ausfallen. Dieses Ergebnis ignorieren Sie dann einfach.

Sind die Controlport-Dongles eingesteckt, wird es zu Störungen der Tastatur kommen, da diese über die selben Leitungen abgefragt wird. Es wird daher empfohlen, erst das Programm zu laden und zu starten und dann während die erste Testseite aufgebaut wird die Dongles einzustecken.

DOC 64 kann von Diskette geladen und dann im RAM gestartet werden, ist aber auch als Eprom auf einer Karte im Expansion-Port lauffähig. Bevor wir auf die allgemeinen Programmfunktionen eingehen, soll daher zunächst die Installation im RAM oder Eprom beschrieben werden. Die einfachste und billigste Methode ist, das Programm einfach von Diskette zu laden und zu starten. Dazu geben Sie ein:

LOAD "DOC 64 $8000",8,8

Zum Start des »Doktors« gibt es drei Möglichkeiten: Mit

SYS 64738

oder einem Reset-Taster wird ein Reset ausgelöst, der das Programm aktiviert, Sie können zum Start nach dem Laden aber auch die Restore-Taste betätigen.

Diese Methode kann versagen, wenn RAMs oder die serielle Schnittstelle oder auch einfach das Diskettenlaufwerk einmal defekt sind. Für diesen Fall empfehlen wir Ihnen, falls Sie die Möglichkeit dazu haben, »DOC 64« auf ein Eprom vom Typ 2764 oder 27128 zu brennen und dann auf einer Expansionport-Karte zu betreiben. Solche Karten (z.B. Typ 9502 von REX Datentechnik) erhalten Sie für ein paar Mark im Fachhandel, sie haben den Vorteil, daß das Programm sofort nach dem Einschalten des Computers ohne Laden zur Verfügung steht.

Brennen Sie das unveränderte File "DOC 64 $8000" (Startadresse: $8000) mit einem geeigneten Eprom-Brenner, zum Beispiel dem »Tiny-Eprommer« aus 64'er 8/88, wie in der Anleitung des Brenners beschrieben auf ein Eprom. Bei Verwendung des Typs 27128 achten Sie darauf, daß das File in beiden 8 kByte-Banks des Bausteins liegt, damit es auch sicher von der Karte im GAME-Bereich des Computers ($8000-$9fff) eingeblendet wird. Nach einer Kontrolle des Eprom-Inhalts stecken Sie dieses in die Karte, schalten sie ein und stecken sie bei ausgeschaltetem C 64 in den Expansion-Port. Wenn Sie jetzt den Rechner einschalten, startet das Programm automatisch.

Bitte enternen Sie vor dem Test alle Soft- und vor allem Hardware-Erweiterungen vom Computer, insbesondere geänderte Betriebssysteme, Floppy-Speeder, Steckkarten in User- oder Expansion-Port (Ausnahme: Die Steckkarte, auf der sich »DOC 64« befindet), sowie Joysticks, Paddles oder sonstige Geräte in den Control-Ports. Ggf. sollten Sie an den noch ausgeschalteten Computer die Dongles an User- und Cassettenport anschließen und erst dann den Rechner einschalten und das Programm laden bzw. starten. Achten Sie darauf, daß Ober- und Unterseite der Dongles nicht verwechselt werden, denn wir wollen den C 64 ja testen, nicht zerstören. Am besten ist es, einen Aufkleber »oben« und »unten« auf die Stecker zu kleben. Da die Tastatur nicht mehr funktioniert, während das Control-Port-Dongle steckt, müssen Sie, falls Sie DOC 64 von Diskette laden, erst das Programm einlesen und wie angegeben starten, und dann die beiden Stecker des Dongles in die Joysticks stecken (sie können vertauscht werden).

Vor der Beschreibung der einzelnen Tests zunächst noch eine Erklärung der allgemeinen Eigenschaften. Der C 64 wird in 33 Schritten getestet, die auf zwei Bildschirmseiten untergebracht sind. Zwischen beiden Seiten wird nach Aufforderung mit der Leertaste umgeschaltet. Die Benutzerführung erfolgt auf Englisch. Unter der Titelzeile mit Programmname, Versionsnummer, Autor und Copyright-Vermerk stehen links die im Test befindlichen Komponenten. Rechts daneben wird nach dem Test der Vermerk »passed« in grün eingetragen, wenn

der Test positiv verlief, oder ein rotes »FAILED« (durchgefallen) bei erkanntem Fehler.

In einigen Fällen ist rechts daneben ein Kommentar zu sehen.

Nachdem alle Tests abgeschlossen sind, startet das Programm nach Tastendruck neu. So lassen sich auch Dauertests durchführen. Beenden können Sie »DOC 64« in jedem Fall nur durch Abschalten des Rechners oder ggf. der Modulplatine, da ein Reset den Doktor ebenfalls neu startet. Bitte entfernen Sie, so vorhanden, die Karte aus dem Expansion-Port nur bei ausgeschaltetem Rechner!

Anmerkung: Falls beim RAM-Test Fehler gemeldet wurden, ist es aus technischen Gründen nicht ausgeschlossen, daß »DOC 64« bei anderen Tests Alarm gibt, obwohl das entsprechende Teil einwandfrei funktioniert. Dies liegt daran, daß die meisten Testroutinen aus verständlichen Gründen nicht ohne RAM auskommen, und daher falsch funktionieren, falls das RAM nicht oder falsch arbeitet.

Da einige Tests in Form von Schleifen durchgeführt werden, oder eine Funktion über längere Zeit überwachen, dauert der gesamte Test einige Zeit. Seite 1 wird in ca. 37 Sekunden abgehandelt (am rechenintensivsten ist dabei der RAM-Test), bei Seite 2 sind es je nach Anzahl der Versuche mindestens 18 Sekunden. Für den Gesamtablauf von »DOC 64« ist also mit etwa 55 Sekunden zu rechnen.

Kommen wir nun zu den einzelnen Tests. Beschrieben werden die Funktionen des Computers, die getestet werden, der Baustein, der bei negativem Test möglicherweise defekt ist (Typenbezeichnung und Position auf der Platine eines C64), das genaue Testverfahren und ggf. die Anzahl der Versuche, die der Test bei negativem Ergebnis durchgeführt wird, bis die »FAILED«-Meldung ausgegeben wird. Durch diese Toleranz werden Fehler nicht aufgeführt, die nur einmalig aufgrund von Hardware-Toleranzen auftraten. Wenn nichts anderes angegeben ist, wird der Test nur einmal durchgeführt.

Ansonsten sind alle Tests aber sehr »streng« und erlauben, wenn überhaupt, nur Toleranzen im Bereich von weniger als einem Prozent. Genauere Hinweise finden sich unten.

Bei den neuesten Modellen des C 64 (sog. »Aldi-C 64«) nahm Commodore umfangreiche Änderungen an der Platine vor, so daß bei diesen Modelle u.U. die Chip-Bezeichnungen nicht mehr stimmen. Der Test müßte aber auf allen Modellen eines voll funktionsfähigen C 64 einwandfrei laufen. Der Autor ist aber für Hinweise etwa auf Hardware-Toleranzen oder geänderte Hardware-Eigenschaften der verschiedenen C 64-Serien dankbar, die zu einem falschen Testergebnis führen.

Auf dem C 128 ist der Test in jedem Fall nur im 64'er Modus funktionsfähig. Interessant wäre, zu prüfen, wie sich beispielsweise C 64-Emulatoren für »größere« Computer wie den Amiga im Test verhalten.

1. Kernal-ROM. Zum Test des Betriebssystem-ROMs werden die Inhalte der Speicherzellen dieses Bausteins ($E000-$FFFF) aufaddiert und die in einem Byte gespeicherte Summe mit dem Sollwert verglichen. Das Programm berücksichtigt, daß es beim C 64 zwei verbreitete Kernal-Versionen gibt, die sich unter anderem darin unterscheiden, daß beim Löschen des Bildschirms verschiedene Default-Zeichenfarben gesetzt werden (vgl. 64'er Sonderheft 33 Seite 116 bzw. 64'er Sonderheft 38 Seite 138). Anhand der Versionsnummer (steht in Adresse 65408) wird geprüft, welche Version vorliegt. Bei Version

0 lautet die Prüfsumme 11, bei Version 3 ist es die 10. Die Versionsnummer wird vom Programm hinter der Prüfmeldung mit angezeigt. Wurde eine andere Versionsnummer als 0 oder 3 gefunden, ergibt der Test immer »FAILED«, vor der Versions-Anzeige stehen dann drei Fragezeichen. Aus diesem Grunde sollten Sie vor dem Test, falls vorhanden, alle Erweiterungen entfernen. Sollte der Test trotz korrekter Seriennummer negativ ausfallen, überprüfen Sie, ob im Steckplatz U4 auf der Computerplatine das Original Kernal-ROM (Typ 2364A, Seriennummer 901227-01) sitzt. Wenn ja, ist dieses defekt. Sollte trotz Original-ROMs eine andere Versionsnummer als 0 oder 3 erscheinen (wäre sehr interessant!), setzen Sie sich bitte mit dem Programmautor in Verbindung.

2. Basic-ROM. Hier werden die Bytes des Basic-ROMs ($A000-$BFFF) addiert und mit dem Sollwert 86 verglichen. Ergibt der Test »FAILED«, ist das Basic-ROM (Typ 2364A, Seriennummer 901226-01, Steckplatz U3) im Computer defekt.

3. Font-ROM. Jetzt werden die Bytes des Zeichensatz-ROMs ($D000-$DFFF) addiert und mit dem Sollwert 248 verglichen. Ergibt der Test »FAILED«, ist das Character-ROM (2332A, 901225-01, U5) im Computer defekt.

4. RAM Uxx. In dieser Stufe werden die RAM-Speicherzellen des 64k umfassenen Speichers auf Funktionsfähigkeit geprüft. xx zeigt dabei den Steckplatz auf der C 64-Platine an (xx=9-12 oder 21-24). Das RAM ist in acht dynamischen RAM-Bausteinen vom Typ 4164 (64 kBit) untergebracht und bit-weise organisiert, d.h., U21 speichert Bit 0 (Wert: 1) aller 65536 Speicherzellen, U9 speichert Bit 1 (Wert: 2), U22 speichert Bit 2 (Wert: 4), U10 speichert Bit 3 (Wert: 8), und so weiter bis U12, welches Bit 7 (Wert: 128) speichert. »DOC 64« speichert in alle Speicherzellen erst den Wert $FF und prüft dann, ob das in dem momentan auf dem Prüfstand befindlichen Chip gespeicherte Bit gesetzt ist. Danach wird jedes Byte mit 0 beschrieben, darauf folgt ein Test, ob das untersuchte Bit gelöscht wird. Der Bereich $0002-$00ff wird von »DOC 64« direkt geprüft, der Bereich $0200-$FFFF von einer im Stack ab $0180 untergebrachten Routine. »DOC 64« prüft demnach nicht die Speicherzellen 0 und 1, die beim C 64 nicht als RAM verwendet werden können (Prozessor-Port), der Bereich $0100-$01ff wird erst später bei der Stack-Prüfung unter die Lupe genommen. Sollte sich herausstellen, daß ein Bit nicht gesetzt oder gelöscht werden kann, zeigt das Programm dies durch die Meldung »FAILED« neben dem vermutlich defekten RAM-Chip an. Dahinter erscheint noch die Adresse, an der der Fehler auftrat, in hexadezimaler Darstellung. Die RAMs werden nacheinander unabhängig voneinander getestet. Sollten mehrere RAM-Chips defekt sein, erscheint die Meldung »FAILED« bei ALLEN defekten Bausteinen. So erkennen Sie im Falle eines Falles gleich, welchen Chip Sie auswechseln sollten.

Auf dem C64 II stimmen die Chip-Bezeichnungen nicht, da hier das RAM in einem einzigen IC untergebracht ist.

5. Color-RAM. Für den Farbspeicher steht ein eigener statischer RAM zur Verfügung, der daher auch extra getestet werden muß. In die 1024 Speicherzellen, die je vier Bits speichern, wird nacheinander der Wert 10 und 5 (abwechselnd gesetztes und gelöschtes Bit) geschrieben und durch Auslesen kontrolliert. Da die vier höherwertigen Bits (höheres Nibble) nicht vorhanden sind, erübrigt sich ein Test. Wird ein Fehler festgestellt, zeigt dies die Meldung »FAILED« an. Dann sollten Sie den Farbram-Chip auf dem Steckplatz U6 durch einen neuen vom Typ 2114-2 austauschen.

6. ALU. Diese Abkürzung steht für »Arithmetical Logical Unit« und meint den Teil des Prozessors 6510, der für das Rechnen zuständig ist. Durch recht

komplizierte Rechenoperationen werden die Befehle ADC, SBC (im Dezimal- und Binärmodus), AND, ORA, EOR, LSR, ASL, ROR und ROL getestet. Tritt hier ein Fehler im Ergebnis auf, kann der C 64 nicht mehr rechnen. In diesem sicher nur sehr selten auftretenden Fall muß der Prozessor 6510 (U7) gegen einen neuen ausgewechselt werden.

7. JMP ($xxff). Bei diesem Test wird auch bei Ihrem Rechner mit ziemlicher Sicherheit »FAILED« ausgegeben. Warum? Der verwendete Prozessor 6510 enthält einen Maskierungsfehler, durch den indirekte Sprünge in Maschinensprache dann nicht richtig ausgeführt werden, wenn das Lowbyte der Speicherzelle mit der Sprungadresse den Wert $ff hat. Beispiel: Kommt in einem Maschinenprogramm der Befehl JMP ($53ff) vor, holt sich der Prozessor völlig zu Recht das Lowbyte der neuen Sprungadresse aus $53ff. Bei der Ermittlung des Highbytes »vergißt« er jedoch, den in diesem Fall auftretenden Übertrag zu berücksichtigen und holt das Lowbyte nicht aus $5400, sondern aus $5300. »DOC 64« prüft, ob auch bei Ihrem C 64 dieser Fehler vorhanden ist. Dazu wird ab $ceff ein Sprung in eine Routine von »DOC 64« generiert, an $cf00 steht das Highbyte der ersten Routine, an $ce00 das der zweiten. Die beiden Routinen beginnen an Adressen, die das selbe Lowbyte haben (steht an $ceff). Je nachdem, welche Routine der Prozessor ausführt gibt »DOC 64« entweder »passed« oder »FAILED« aus. Sorgen Sie sich nicht, wenn »FAILED« erscheint, es ist nicht notwendig, den Prozessor auszuwechseln, zumal auch ein neuer 6510 diesen Fehler hat. Sie sollten jedoch, falls Sie in Maschinensprache programmieren, den Befehl JMP ($xxxx) vermeiden, wenn nicht sichergestellt ist, daß das Lowbyte nicht an einer Adresse $xxff steht. Für reine Basic-Programmierer ist dieser Mangel nicht von Bedeutung.

8. Stack. Da zuvor beim RAM-Test der Bereich $0100-$01ff (Stack) noch ausgespart wurde, erfolgt die Prüfung jetzt. Dazu legt das Programm einen ungefähr 252 Bytes lagen Teil von sich selbst (ab $8000) auf den Stack (Befehl PHA), und holt ihn anschließend mit PLA wieder herunter. Dabei erfolgt ein Vergleich mit dem Original. Sollte hier ein (sehr unwahrscheinlicher) Fehler auftreten, ist der Prozessor defekt, falls zuvor keine RAM-Fehler erkannt wurden. Möglicherweise liegt auch ein Defekt auf der Platine (Bus) vor.

9. SID env. Auf der zweiten Bildschirmseite wird der Hüllkurvengenerator der Stimme 3 des Soundchips SID einer Prüfung unterzogen. Das Programm erzeugt einen Ton der Wellenform »Dreieck« und mißt im Interrupt die Dauer der Release-Phase (Register 54300). Beträgt die Anzahl der Interrupts auch nach dem 10. Versuch nicht genau 83, oder arbeitet der SID gar nicht, gibt »DOC 64« die Meldung »FAILED« aus. Dann sollten Sie einen neuen SID (6581, U18) in Erwägung ziehen, falls auch bei anderen Programmen Tonstörungen zu erkennen sind.

Die Toleranzgrenzen dieser Prüfung sind bewußt sehr niedrig eingestellt, um wirklich jeden Fehler zu finden. Es ist daher nicht immer ausgeschlossen, daß DOC 64 hier aufgrund von hardwarebedingten Toleranzen Fehlalarm auslöst, obwohl der Baustein technisch einwandfrei arbeitet.

10. SID osc. Nach dem Hüllkurvengenerator wird noch der Oszillator, der eigentliche Tongenerator der Stimme 3 einer Prüfung unterzogen. Das Programm generiert Töne in den Wellenformen »Rechteck«, »Sägezahn« und »Dreieck« und liest den Tongenerator über das Register 54299 aus. Bei »Rechteck« dürfen hier nur die Werte 255 und 0 auftreten. Bei »Sägezahn« bildet das Programm die korrekte Kurve (perioisch von 0 bis 255 ansteigend) künstlich nach und vergleicht sie mit dem tatsächlichen Ergebnis. Bei »Dreieck« (periodisch von 0 bis 255 an- und dann wieder auf 0 absteigend) darf die Differenz zwischen

zwei benachbarten Meßwerten nur -2, -1, 1 oder 2 sein. Andernfalls erscheint nach 20 Versuchen die Fehlermeldung, und der SID 6581 (U18) sollte ausgewechselt werden.

11. SID Data. Die ersten 24 Register des SID sind »Write-Only-Register«. In sie kann zur Tonsteuerung ein Wert geschrieben werden, ein Auslesen der Speicherzellen 54272 bis 54296 führt bei einem intakten SID immer zum Ergebnis Null. Dies wird hier geprüft. Tritt ein von Null verschiedener Wert auf, stimmt etwas mit dem Soundchip nicht (U18, 6581).

12. A/D Conv. Hier werden die A/D-Wandler des SID getestet. Dazu muß der Dongle mindestens am Controlport 1 angeschlossen sein. »DOC 64« untersucht nur die Speicherzellen 54297 und 54298, die den Wert 24-25 oder 45-46 (Toleranz!) enthalten müssen. Stimmt etwas nicht, sollten Sie den Soundchip (U18) gegen einen neuen vom Typ 6581 auswechseln, da er die Wandler enthält. Aber auch ein Fehler in der SID-Beschaltung ist denkbar. Die A/D-Anschlüsse an Port 2 werden bewußt nicht geprüft, da auch sie intern zum SID laufen. Fehlt der Dongle, meldet der Test »FAILED«.

Diese A/D-Wandler sind leider ziemlich ungenau. Es ist daher nicht ausgeschlossen, daß in Einzelfällen bei einigen Lesern die FAILED-Meldung erscheint, obwohl der Wandler korrekt arbeitet. Der Grund: Der hier eingebaute SID liefert andere Werte als der des Programmautors. Auch Toleranzen bei den verwendeten Widerständen können die Ursache sein. Mit folgenden Befehlen läßt sich, wenn erwünscht, das Programm "eichen" (bitte bei angeschlossenen Dongles eingeben, auch wenn dies wegen der Tastatur-Beeinflusung schwerfallen mag):

```
POKE 35744,PEEK(54297)
POKE 35755,PEEK(54298)
```

13. CTRL-Ports. Hier werden die beiden Joystickports getestet. Dazu muß der entsprechende Dongle an beiden Ports angeschlossen sein. Die fünf Stick-Leitungen (vier Richtungen und Feuer) werden erst an Port 1 auf Ausgang und an Port 2 auf Eingang geschaltet. Dann übermittelt das Programm auf diesem Weg von Port 1 zu Port 2 einige Testdaten, die originalgetreu ankommen müssen. Danach wird Port 2 auf Ausgang und Port 1 auf Eingang geschaltet und die Prozedur umgekehrt wiederholt. Tritt »FAILED« auf, tauschen Sie die CIA 1 (U1) gegen einen neuen Baustein vom Typ 6526 aus.

14. Cass.Port. Obwohl selten für den eigentlichen Zweck (Datasette) benutzt, kann der Hardware-Bastler auch mit dem Cassettenport einiges anfangen. Hier wird er mit Hilfe des Dongles getestet. Dazu setzt der Computer die Motor-Leitung auf bestimmte Prüfwerte, die über den Dongle an die Sense-Leitung weitergegeben werden. Über Speicherzelle 1 erfolgt die Prüfung. Stellt das Programm Unstimmigkeiten fest, fehlt entweder das Dongle, oder der Prozessor (6510, U7) oder einer der Transistoren in der Port-Beschaltung (Q1 bis Q3) hat sich verabschiedet.

15. serial Port. Zum Test der Disketten-Schnittstelle wird kein Dongle benötigt, da über ein Gatter-IC bereits wichtige Leitungen der Schnittstelle verbunden sind. Auch hier wird wieder wechselseitig durch Setzen bestimmter Werte und Prüfung, ob sie am Port wieder ankommen die Funktionsfähigkeit unter die Lupe genommen. Bei »FAILED« können Sie davon ausgehen, daß entweder die CIA 2 (6526, U2) oder das Gatter-IC (U8, Typ 7406N bzw. M53206P) defekt ist.

16. Userport. »DOC 64« prüft die acht frei definierbaren I/O-Pins des

Userports (Port B), die gewöhnlich nicht benutzt werden (höchstens für die Kommunikation mit einer RS 232-Schnittstelle, mit einem parallel angeschlossenen Drucker oder Eigenentwicklungen), sowie weitere Userport-Funktionen. Auch hierzu muß der Dongle vorhanden sein, sonst wird »FAILED« angezeigt. Zunächst werden die Pins PA2 und PB7 sowie PB0 und PB5 zusammengeschaltet. Je ein Anschluß der Paare dient als Ausgang, der andere als Eingang. Ähnlich wie oben beim Joystickport erfolgt der Test auch hier mit Test-Bits. Zum Test des ATN-Bits wird dieses mit PB6 verschaltet. DOC 64 setzt PB6 kurzzeitig auf High und prüft, ob der C 64 einen NMI auslöst (ATN-Leitung). Die nächsten Tests betreffen die Schieberegister, von denen je eines in jeder CIA eingebaut ist. Dazu werden die Steuerleitungen PB1 bis PB4 mit SP1, SP2 und CNT1, CNT2 verbunden. Nacheinander wird jetzt ein Testwert seriell in die Schieberegister geschoben, der dort auch ankommen muß. Als letzter Test werden im Dongle die Pins FLAG und PC2 verbunden. Über PC2 kommt ein kurzer Impuls, wenn DOC 64 einen Schreib- oder Lesezugriff auf PB ausführt. Über eine NMI-Routine, die auf FLAG reagiert, wird dies getestet. Tritt ein Fehler auf, ist mit Sicherheit eine CIA (U1 oder U2, Typ 6526) nicht in Ordnung.

17. t/Raster. Eine Messung der Zeit t, die der Videochip zum Aufbau einer Rasterzeile (in diesem Fall Zeile 100) benötigt, erfolgt. Dazu dient ein Zähler, der während dem Aufbau der Zeile von 0 bis genau 7 zählen muß. 20 Versuche sind zugelassen, dann wird bei Mißerfolg »FAILED« ausgegeben. Dann ist sicher der Videochip (6569, U19) defekt, da dieser ja auch den Prozessortakt erzeugt und somit die Geschwindigkeit des Zählers vorgibt.

18. Sprites. Die Fähigkeit des VICs, Sprites darzustellen, wird folgendermaßen auf die Probe gestellt: »DOC 64« stellt im unteren Bereich des Bildschirms drei Sprites dar (daher das leichte Flimmern an dieser Stelle beim Test) und prüft mit Hilfe der Vergrößerungs-Register und der Sprite-Sprite-Kollisionserkennung, wann sich zwei Sprites überlappen. Anschließend wird für ein Sprite die Kollision mit einem Zeichen des Textbildschirms geprobt. Tritt eine Störung auf, meldet »DOC 64« dies. Dann kann nur der Videochip (6569, U19) defekt sein, falls zuvor keine RAM-Störungen gemeldet wurden.

19. Raster-IRQ. Nun muß noch die Fähigkeit des VIC unter die Lupe genommen werden, IRQs auszulösen. Dazu wird ein Raster-IRQ bei Rasterzeile 100 programmiert. Erfolgt dieser, liest »DOC 64« das Raster-Register aus und überprüft, ob auch wirklich Zeile 100 den Interrupt ausgelöst hat. Erfolgt der Raster-IRQ nicht innerhalb einer vorgegebenen Toleranzzeit (Zählung bis 65536), wurde der Test ebenfalls nicht bestanden. Es stehen 20 Versuche zur Verfügung. Erst nach deren Ablauf erscheint »FAILED«. In diesem Fall ist wahrscheinlich der VIC (6569, U19) in die ewigen Jagdgründe übergegangen, oder die Verbindungsleitung zwischen Prozessor und VIC ist defekt.

20. Timer-NMI. Die CIA 2 ($DD00) wird so geschaltet, daß ihr Timer A von 65535 auf 0 zählt und dann einen NMI auslöst. Dies wird im Hauptprogramm geprüft. Dazu wird ein weiterer Zähler verwendet, der exakt den Wert 5376 erreichen muß, wenn der NMI erfolgt. Dieser Test hat drei Versuche. Verläuft er negativ, liegt ein Defekt bei CIA 2 (6525, U2), in der NMI-Leitung oder beim Prozessor (6510, U7) vor.

21. TOD-NMI. Nun wird noch die Möglichkeit der Echtzeituhr (Time of day, TOD) der CIA 2 ($DD00) geklärt, bei Erreichen der Alarmzeit einen NMI auszulösen. Die TOD wird auf 00:00:00:0 Uhr gestellt und gestartet, die Alarmzeit wird auf 00:00:02:0 gestellt, also nach zwei Sekunden. Trat auch nach drei Versuchen der NMI nicht genau nach zwei Sekunden auf (dazu wird

ein Abwärtszähler verwendet, der von 196608 bis 65536 zählt), erscheint »FAILED«. Dann ist wieder entweder die CIA 2 (6525, U2), die NMI-Leitung oder der Prozessor (6510, U7) kaputt.

22. Timer-IRQ. Hier wird die Langzeitkonstanz des System-IRQ (ausgelöst durch Timer A der CIA 1 ($DC00)) getestet. Das Programm mißt die Anzahl der aufgetretenen IRQs, während ein Zähler bis 720896 zählt. Es müssen genau 234 IRQs sein. Stimmt dies nicht (drei Versuche), liegt ein Defekt in der CIA 1 (U1, 6526) vor, oder in der IRQ-Leitung oder auch im Prozessor (6510, U7).

23. Timer CIA1. Dieser Programmpunkt testet, ob die beider Timer der CIA 1 richtig arbeiten. Dazu erhalten beide den selben Startwert (51200) und zählen dann vom Systemtakt getriggert abwärts. Hat Timer A die Null erreicht, prüft »DOC 64«, ob dies auch für Timer B der Fall ist. Wenn nicht, liegt mit Sicherheit ein Defekt in der CIA 1 (6526, U1) vor.

24. Timer CIA2. Dieser Programmpunkt testet entsprechend, ob die beider Timer der CIA 2 richtig arbeiten. Dazu erhalten beide den selben Startwert (51200) und zählen dann vom Systemtakt getriggert abwärts. Hat Timer A die Null erreicht, prüft »DOC 64«, ob dies auch für Timer B der Fall ist. Wenn nicht, liegt mit Sicherheit ein Defekt in der CIA 2 (6526, U2) vor.

25. TOD CIA1. Jetzt wird noch die Ganggenauigkeit der Echtzeituhr der CIA 1 geprüft. Zur Messung wird die CIA 2 verwendet. Timer A der CIA 2 wird so geschaltet, daß er ununterbrochen Systemtakte von 9850 abwärts zäht. Bei einer Taktfrequenz von etwa 0,985 MHz bedeutet dies, daß Timer A alle 1/100 Sekunde einen Unterlauf produziert. Timer B der CIA 2 zählt die Underflows von Timer A. Am Stand von Timer B kann also die abgelaufene Zeit in 1/100 Sekunden abgelesen werden. Die Uhr der CIA 1 wird auf Null gestellt. Jetzt wartet das Programm, bis sie 5 Sekunden erreicht hat. Timer B wird ausgelesen, er muß genau 2500 oder 2501 (Toleranz) Zyklen hinter sich gebracht haben. Stimmt dies auch nach drei Versuchen nicht, liegt ein Defekt in der CIA 1 (6526, U1), der CIA 2 (6526, U2), im Systemtakt oder der 50 Hz Netzfrequenz vor.

26. TOD CIA2. Zum Schluß wird noch die Ganggenauigkeit der Echtzeituhr der CIA 2 geprüft. Timer A der CIA 2 wird so geschaltet, daß er ununterbrochen Systemtakte von 9850 abwärts zäht. Timer B der CIA 2 zählt die Underflows von Timer A. Am Stand von Timer B kann also die abgelaufene Zeit in 1/100 Sekunden abgelesen werden. Die Uhr der CIA 2 wird auf Null gestellt. Jetzt wartet das Programm, bis sie 5 Sekunden erreicht hat. Timer B wird ausgelesen, er muß genau 2500 oder 2501 (Toleranz) Zyklen hinter sich gebracht haben. Stimmt dies auch nach drei Versuchen nicht, liegt ein Defekt in der CIA 2 (6526, U2), im Systemtakt oder der 50 Hz Netzfrequenz vor.

Danach ist das Testverfahren abgeschlossen. Es kann auf Tastendruck (Leertaste) neu gestartet werden.

Falls Sie sich für die genaue interne Funktionsweise von »DOC 64« interessieren, oder gar Erweiterungen oder Änderungen vornehmen wollen, wird die unten angegebene Speicherbelegung nützlich sein. So, und nun wünschen wir Ihnen noch viel Erfolg bei Umgang mit »DOC 64«. Auß daß das Programm möglichst selten Fehler registriert, und daß Sie diese im Zweifelsfalls schnell, billig und einfach reparieren (lassen) können! Sollten Sie noch Fragen oder Anregungen haben, steht der Autor gerne zur Verfügung (bitte Rückporto beilegen).

Speicherbelegung für »DOC 64« in der Version 2.2 (hexadezimal):

```
0002-0003 IRQ-Zähler
0004      Prüfbyte
0005      RAM-Nummer
00b5-00b6 temporärer Zeiger
00f7      Anzahl der Versuche
00f8      letzter SID-Oszillator-Wert
00f9      Versuchszähler
0100-01ff Stack
0180-01c5 RAM-Test Unterroutine
01b7      Zweierpotenzen
8000-8fff Programm »DOC 64«
8000-8003 Reset/Restore-Vektoren
8004-8008 CBM80-Kennung
8009      Reset-Routine/Start Test
ce00      »Falle«: falsches Highbyte für JMP ($ceff)
ce10      falsches Sprungziel für JMP ($ceff)
ceff-cf00 korrekte Parameter für JMP ($ceff)
cf00-cfff SID-Oszillatorspeicher
cf10-cf12 korrektes Sprungziel für JMP ($ceff)
```

Kalender unter Kontrolle

Mit einem kleinen Trick bekommen Sie als Programmierer die Jahrhunderte in den Griff. Wir zeigen Ihnen, mit welchen Verfahren Routinen geschrieben werden, die mit Tagesdaten umgehen.

In vielen Computerprogrammen kommen als Daten auch Tage vor. Beispielsweise wird ein Datum eingegeben, das dann im Folgenden weiterverwertet wird. Nun könnte das Programm natürlich beispielsweise den Wochentag dieses Datums ausrechnen. Dennoch werden Sie eine solche Funktion nur bei sehr wenigen Programmen finden. So kompliziert kann die Berechnung doch nicht sein. Doch wie sagen wir dem Rechner etwa, daß der Dezember 31 Tage und der Februar manchmal 28, manchmal aber auch 29 hat? Am besten gar nicht.

Manchmal muß auch die Differenz zweier Daten in Tagen berechnet werden. Wozu man das braucht? Denken Sie nur an Biorhythmus-Programme (Duden!). Aber auch für viele andere ernsthafte Bereiche, etwa das weite Feld der »Bürokratie« werden solche Algorithmen benötigt. Wie gesagt, nur wenige Lösungen sind bekannt. Ein Beispiel dafür, daß das Problem zu schwer ist? Mag sein. Aber es gibt einige Tricks, wie man sich das Leben hier bedeutend einfacher machen kann.

Stellen wir uns eine Aufgabe. Ein Programm soll die Nummer eines Tages im Kalenderjahr ausgeben. Wir sollten jetzt nicht einfach wild drauflos programmieren, sondern das Problem systematisch angehen. Zunächst einmal ist das gesuchte Datum einzugeben.

```
10 INPUT"DATUM FORMAT TTMMJJJJ ";D$
20 T=VAL(MID$(D$,1,2))
30 M=VAL(MID$(D$,3,2))
```

```
40 J=VAL(MID$(D$,5))
```

Wir geben den String im Format »TTMMJJJJ« ein, also ohne Punkte oder sonstige Trennzeichen. Die Eingabe »18021991« <RETURN> bedeutet mithin 18. Februar 1991. Das Jahrhundert soll also eingegeben werden. In einem anwenderfreundlichen Programm erscheinen diese Hinweise übrigens auch am Bildschirm, das aber nur nebenbei. Danach zerlegen wir die Eingabe in drei Teile und ermitteln die Wertigkeit von Tag, Monat und Jahr. In unserem Beispiel wird T = 18, M = 2 und J = 1991. Zur Sicherheit kann eine Prüfung der Eingabe auf Plausibilität nicht schaden, das erledigen die folgenden Zeilen:

```
50 IF M<1 OR M>12 THEN RUN
60 IF T<1 OR T>31 THEN RUN
```

Wie berechnet man jetzt die Tagesnummer? Im Prinzip könnte man einfach durchzählen. Aber es geht einfacher. In der folgenden Tabelle sehen Sie zusammengefaßt die Anzahl Tage der Monate März bis Dezember:

Monat	Tage	Monat	Tage
3	31	8	31
4	30	9	30
5	31	10	31
6	30	11	30
7	31	12	31

Bis auf den Sprung von 7 nach 8 wechseln sich also immer die Werte 30 und 31. Und nun probieren Sie im Direktmodus folgende Schleife aus:

```
FOR M=3 TO 12 : PRINT M, INT (30.6 * (M+1) ) : NEXT
```

Dieser Algorithmus gibt also genau die obige Tabelle aus. Das ist einfacher, als die Monatslängen in einem Array zu Programmbeginn zu definieren und kostet weniger wertvollen Speicherplatz. Aber wie steht es mit Januar und Februar? Das ist weiter nicht schlimmt, wir beginnen die Zählung des Jahres einfach mit dem ersten März und hängen die Monate Januar und Februar hinten nach dem Dezember an. Der Februar ist somit der letzte Monat unseres »EDV-Jahres«, es ist egal, ob er 28 oder 29 Tage hat, die Zählung arbeitet immer richtig.

Verfollständigen wir unser Programm nach dieser Methode:

```
70 REM WELCHE TAGESNUMMER ?
80 IF M>2 THEN N=INT(30.6*(M+1))-63+T:GOTO 100
90 N=INT(30.6*(M+13))-428+T
100 PRINT"NUMMER"N
110 END
```

In Zeile 80 erfolgt eine Fallunterscheidung. Ist der eingegebene Monat März oder höher, kann die angegebene Formel verwendet werden. Wir ziehen vom Ergebnis noch 63, die Nummer des ersten März ab. Ebenso muß in Zeile 90 bei der Bearbeitung der Monate Januar und Februar das Ergebnis kosmetisch nachbehandelt werden.

Diese Routine funktioniert jetzt aber nur für ein Jahr (und produziert unter Umständen bei Schaltjahren falsche Werte). In der Praxis wird man diese Routine wohl kaum benötigen. Das folgende Problem tritt dagegen weitaus häufiger auf. Beispielsweise wird die Frage gestellt, wie viele Tage

zwischen zwei bestimmten Daten liegen. Kennen Sie Ihr genaues Alter in Tagen? Man geht hierbei so vor, daß jedes Datum in eine absolute Zahl umgewandelt wird. Dazu nimmt man die Anzahl der Tage, die seit einem bestimmten fest definierten Ereignis vergangen sind, etwa seit Christi Geburt, seit dem Geburtstag der Freundin, seit der Vereinigung Deutschlands oder was auch immer. Aus programmtechnischen und astronomischen Gründen wählt man in der Praxis dabei das Datum 1.01.4713 vor unserer Zeitrechnung, ein Datum, das weit vor allen geschichtlich überlieferten Ereignissen liefert. Man nennt dieses Datum auch »julianisches Datum«. Wir müssen jetzt also das in den Variablen T, M und J übergebene Datum in das julianische Datum, eine einfache Zahl, umwandeln. Folgende Unterroutine übernimmt das freundlicherweise für uns:

```
1000 REM KALENDER -> JULIANISCH
1010 IF M>2 THEN M=M-3:GOTO 1030
1020 M=M+9:J=J-1
1030 JH=INT(J/100):JA=J-100*JH
1040 JD=INT(146097*JH/4)+INT(1461*JA/4)
1050 JD=JD+INT((153*M+2)/5)+T
1060 RETURN
```

Die genaue Funktionsweise, insbesondere die Bedeutung der Konstanten 146097, 1461 und 153 kann hier aus Platzgründen leider nicht beschrieben werden, da umfangreiche astronomische Überlegungen dahinterstecken. Kluge Menschen haben sich schon den Kopf darüber zerbrochen, uns soll daher die fertige Lösung genügen. Diese berücksichtigt natürlich auch Schaltjahre und Jahreszahlen. In Zeile 1010 prüfen wir, ob der Monat März oder höher war. Wenn ja, rechnen wir ihn in unseren »EDV-Jahr-Monat« um. In Zeile 1020 erfolgt die Rechnung für die Monate Januar oder Februar. Wir addieren 9 zum Monat und erniedrigen dafür das Jahr um eins. In Zeile 1030 berechnen wir das Jahrhundert (JH) und Kalenderjahr ohne Jahrhundert (JA), dies läßt sich mit der Zerlegung in High- und Lowbyte bei der Binärrechnung vergleichen und wird im folgenden zur Berücksichtigung der Schaltjahr (Zeile 1040) benötigt. In 1050 addieren wir dann noch die nach bekannter Methode ermittelte Tagesnummer im Jahr zu JD. JD enthält dann das julianische Datum zu T, M und J.

Um die Funktionsweise zu testen, löschen Sie bitte die Zeilen 70 bis 110 des vorherigen Musterprogramms und geben folgendes ein:

```
70 GOSUB 1000
80 PRINT "JULIANISCH: "JD
90 END
```

Auch der umgekehrte Weg läßt sich gehen. Wir wandeln das in JD übergebene julianische Datum in das Kalenderdatum um und übergeben dieses in den Basic-Variablen T, M und J. Die Routine dazu soll ab Zeile 2000 beginnen und sieht so aus:

```
2000 REM JULIANISCH -> KALENDER
2005 J=INT((4*JD-1)/146097)
2010 JD=4*JD-1-146097*J
2020 T=INT(JD/4)
2030 JD=INT((4*T+3)/1461)
2040 T=4*T+3-1461*JD
2050 T=INT((T+4)/4)
2060 M=INT((5*T-3)/153)
2070 T=5*T-3-153*M
```

```
2080 T=INT((T+5)/5)
2090 J=100*J+JD
2100 IFM<10THENM=INT(M+3):GOTO2120
2110 M=INT(M-9):J=J+1
2120 RETURN
```

Man sieht also, daß der umgekehrte Weg viel umständlicher ist komplizierter ist. Wir werden aber später sehen, daß er durchaus nützlich sein kann, etwa wenn es um Fristen geht. Mit folgendem Programm probieren wir die Funktionsweise der Routine aus:

```
10 INPUT "JULIANISCHES DATUM ";JD
20 GOSUB 2000
30 PRINT "KALENDERDATUM:";T;".";M;".";J
40 END
```

Unsere Aufgabe war, die Differenz zwischen zwei Daten in Tagen zu errechnen. Dazu dient folgendes Programm, das auf die Routine ab 1000 zurückgreift:

```
10 INPUT"DATUM 1 FORMAT TTMMJJJJ ";D$
20 T=VAL(MID$(D$,1,2))
30 M=VAL(MID$(D$,3,2))
40 J=VAL(MID$(D$,5))
50 GOSUB 1000
60 J1=JD
70 INPUT"DATUM 2 FORMAT TTMMJJJJ ";D$
80 T=VAL(MID$(D$,1,2))
90 M=VAL(MID$(D$,3,2))
100 J=VAL(MID$(D$,5))
110 GOSUB 1000
120 R=JD-J1
130 PRINT "DAZWISCHEN LIEGEN"R"TAGE."
140 END
```

In Zeilen 10 bis 40 lassen wir das erste Datum eingeben und werten es aus. In Zeile 50 wird die Berechnung des julianischen Datums aufgerufen. Das Unterprogramm gibt den Wert in JD zurück, diese Variable müssen wir zur weiteren Berechnung in J1 zwischenspeichern (Zeile 60). Danach werten wir auf gleiche Weise in 70 bis 110 das zweite Datum aus und berechnen in Zeile 120 die Differenz. Geben Sie in das Programm zum Spaß einmal als erstes Ihr Geburtsdatum ein, und als zweites das heutige Datum. Sie wissen dann Ihr Lebensalter - in Tagen! Diese Zahl könnte nun beispielsweise mit SIN-Funktionen für einen Biorhythmus aufbereitet werden, doch das soll nicht Inhalt dieses Aufsatzes sein.

Wir können nun die Differenz zwischen zwei Daten (Mehrzahl von Datum) berechnen (lassen). Ein anderes Programm muß mit Tagesfristen operieren können. Wann muß der Antrag, der Einspruch, für den man bekanntlich 30 Tage nach Zugang Zeit hat, beim Finanzamt eingereicht sein? An welchem Tag sollte eine Rechnung mit Zahlungsziel »20 Tage nach Rechnungsdatum« spätestens bezahlt werden? Das folgende Programm rechnet es für Sie aus.

```
10 INPUT"DATUM FORMAT TTMMJJJJ ";D$
20 T=VAL(MID$(D$,1,2))
30 M=VAL(MID$(D$,3,2))
40 J=VAL(MID$(D$,5))
50 GOSUB 1000
60 INPUT"FRIST IN TAGEN ";R
```

```
70 JD=JD+R
80 GOSUB 2000
90 PRINT "LAEUFT AM";T;".";M;".";J;". AB!"
100 END
```

Es findet wieder die bekannte Routine zur Eingabe und Zerlegung des Tagesdatums Verwendung. In Zeile 50 wird das julianische Datum dazu berechnet. Wir geben daraufhin die Frist in Tagen ein. Hier sind auch negative Werte und die Null zugelassen! Die Eingabe addieren wir zum julianischen Datum, das ergibt ein neues julianisches Datum. Zeile 80 prüft nun, welches Kalenderdatum diesem entspricht und gibt es in Zeile 90 aus. Probieren Sie einmal aus, ob alles funktioniert.

Jetzt fehlt nur noch ein wichtiger Punkt. Können Sie herausfinden, aus welchen Wochentag ein bestimmtes Datum fällt? Wissen Sie beispielsweise, an welchem Tag Sie geboren wurden? (Sie meinen, das wäre egal? Haben Sie schon einmal den Ausdruck »Sonntagskind« gehört? Fragen Sie einmal einen Astrologen!) Aber ernsthaft: Viele Planungs- und Verwaltungsprogramme könnten wirksam Schaden vermeiden, wenn sie selbständig feststellen, ob Tage auf Samstag, Sonntag oder Feiertag fallen (natürlich nur, wenn ein Verzeichnis aller Feiertage eingebaut ist). Ein Beispiel wäre eine Urlaubsverwaltung.

Die Folge der sieben Tage wiederholt sich ja regelmäßig. Sie brauchen also nur die Anzahl der Tage zu bestimmen, die seit einem bestimmten Datum, etwa dem 1.01.1980 vergangen sind. Wie das geht, haben wir ja schon gelernt. Das Ergebnis wird durch sieben geteilt. Der Rest ist die Nummer des Wochentages. Wir müssen noch wissen, daß der 1.1.80 ein Dienstag war, zu der Formel sollte also noch zwei addiert werden, sonst kommt das Programm auf Dienstag.

```
10 INPUT"DATUM FORMAT TTMMJJJJ ";D$
20 T=VAL(MID$(D$,1,2))
30 M=VAL(MID$(D$,3,2))
40 J=VAL(MID$(D$,5))
50 GOSUB 1000
60 UNT=JD-723121+2
70 TAG=UNT-INT(UNT/7)*7
80 TAG$(0)="SONNTAG"
90 TAG$(1)="MONTAG"
100 TAG$(2)="DIENSTAG"
110 TAG$(3)="MITTWOCH"
120 TAG$(4)="DONNERSTAG"
130 TAG$(5)="FREITAG"
140 TAG$(6)="SAMSTAG"
150 PRINT"WOCHENTAG: "TAG$(TAG)
160 END
```

Bis zu Zeile 60 enthält das Programm nichts neues. In Zeile 60 berechnen wir, wie viele Tage seit dem 1.1.80 (dessen julianisches Datum 723121 ist) vergangen sind. Dazu wird 2 addiert, um die Skala zu normieren. In Zeile 70 findet sich ein kleiner »Kniff«. Da es im Commodore-Basic keine Funktion zur Berechnung des Restes einer Division gibt, behelfen wir uns so:

```
A MOD B = A - INT (A/B) * B
```

A soll durch B geteilt und der Rest ermittelt werden. B ist hier konstant 7 (Anzahl Wochentage), A ist UNT, die Differenz der Tage. Mit Hilfe des in Zeilen 80 bis 140 definierten String-Arrays geben wir in Zeile 150 dann den

Wochentag aus. Man könnte das Ganze noch etwas eleganter mit einer FOR..NEXT und READ..DATA-Struktur gestalten, die die Wochentagsnamen einliest, aber hier geht es nur ums Prinzip. Und nun sollten Sie unbedingt einmal Ihr Geburtsdatum eingeben und den Wochentag feststellen lassen.

Die Formeln, die Ihnen hier vorgestellt wurden, sind auf den ersten Blick sicher sehr schwer verständlich. Es ist aber gar nicht notwendig, daß Sie wissen, wie sie funktionieren. Bauen Sie einfach je nach Bedarf die Unterprogramme ab 1000 und/oder 2000 in Ihr Programm ein, und arbeiten Sie damit.

Großputz - Die Tastatur reinigen

Viele C 64-Anwender haben inzwischen erkannt, daß etwas Hygiene auch beim Computer sein muß (siehe 64'er 2/88). Von Zeit zu Zeit wird beispielsweise der Druckkopf einer gründlichen Reinigung unterzogen, wie es ja auch im 64'er Magazin (2/86 und 2/87) schon beschrieben wurde. Auch der Schreib/Lesekopf des Floppy-Laufwerks kann eine »Ölung« von Zeit zu Zeit ganz gut vertragen. Aber ein Teil der Anlage, das sehr anfällig gegen Schmutz ist, wird dabei oft übersehen: Die Tastatur. Wir wollen Ihnen daher eine ausführliche Beschreibung geben, wie man das Keyboard säubert. Sie sollten diese Schritte immer dann sorgfältig durchführen, wenn Tasten nur noch auf sehr starken Druck oder gar nicht mehr reagieren.

Für den Großputz benötigen wir folgendes: Einen großen und einen kleinen Kreuzschlitz-Schraubenzieher, eine Zange, einen Staubsauger, ein lauwarmes Wasserbad, einen fusselfreien feuchten Lappen, eventuell Desinfektions-Spray, Nitro-Verdünnung und einen Lötkolben. Für den kompletten Vorgang rechnen Sie etwa eine halbe Stunde.

Vor der Reinigung muß der C 64 abgeschaltet und von allen anderen Geräten getrennt werden (Steckverbindungen lösen). Danach kann der »Brotkasten« umgedreht werden. An seiner Unterseite befinden sich drei Kreuzschlitz-Schrauben, die mit einem geeigneten Schraubendreher herausgedreht werden sollen. Bewahren Sie alle Schrauben sorgfältig auf. Achtung: Durch das Öffnen des Gerätes geht ein eventueller Garantieanspruch verloren! Bei entsprechender Sorgfalt kann aber nichts passieren. Jetzt kann der Brotkasten aufgeklappt werden. Sie sehen den 18-adrigen Kabelbaum, über den die Tastatur mit der Computerplatine verbunden ist. Dieser ist mit einem Stecker an der linken Platinenseite angeschlossen. Ziehen Sie vorsichtig diesen Stecker aus seiner Kontaktleite. Die rote Leuchtdiode kann, muß aber nicht unbedingt ausgesteckt werden (rechte Platinenseite). Jetzt kann die ganze Tastatur abgeschraubt werden. Dazu lösen Sie die acht großen Kreuzschlitz-Schrauben in der Oberseite des Brotkastens. Sie sehen ähnlich aus wie die drei Schrauben, die schon den gesamten Computer zusammenhielten. Lösen Sie vorerst noch nicht die 23 kleinen Schrauben, die auf der bräunlich-orangefarbenen Tastaturplatine sitzen!

Die gesamte Tastatur kann nun abgenommen werden. An dieser Stelle könnten

Sie eine neue Tastatur einsetzen (diese gibt es als Ersatzteil zu kaufen), wenn Sie eine haben. Da dies bei den wenigsten Lesern der Fall sein wird, beschreiben wir jetzt die Säuberungs-Aktion. Nehmen Sie den Computer ohne Tastatur und legen ihn beiseite, er wird zunächst nicht mehr benötigt. Entgegen dem weit verbreiteten Gerücht ist es auch beim C 64 wie bei jedem anständigen Computer möglch, die Tastenkappen einzeln abzuziehen, z.B. zur Reinigung. Das sollten Sie jetzt sehr vorsichtig und bitte ohne allzu große Gewaltanwendung tun. Ziehen Sie etwa mit einer Zange die 65 Tastenkappen (nicht die Leertaste) senkrecht nach oben (wie ein Zahnarzt, der Zähne zieht). Hierbei sollten Sie achtgeben, daß zum einen die Tasten nicht zu sehr verkratzt werden (packen Sie sie mit der Zange links und rechts an, damit das auf der Vorderseite aufgedruckte Grafikmuster nicht leidet), und zum anderen nichts abbricht. Das verwendete Plastik ist zwar wirklich sehr stabil, aber Vorsicht ist nun mal besser als Nachsicht. Auch die SHIFT-LOCK-Taste kann auf diese Weise abgenommen werden. Unter jeder Kappe sitzt eine Spiralfeder, die die Taste nach oben drückt. Sammeln Sie die 65 Federn in einem Becher, sie müssen nur leicht abgestaubt werden.

Die beste Methode zur Reinigung der Tastenkappen ist, sie etwa eine Viertel- bis halbe Stunde in einem lauwarmen Wasserbad zu lagern (fließendes Wasser?), damit der Schmutz sich lösen kann. Bitte verwenden Sie keine Chemikalien dazu, sie könnten das Plastik angreifen. Jetzt ziehen Sie auch die Kappe der breiten Leertaste ab. Dazu packen Sie diese mit der Zange oben und unten genau in der Mitte und ziehen sie heraus. Den Metallbügel, der die Taste waagerecht hält, lagern Sie am besten bei den Metallfedern (eine solche steckt auch in der Space-Taste, Sie haben jetzt also 66 davon). Diese Metallteile dürfen nicht feucht gereinigt werden (Rostgefahr). Die Leertaste legen Sie ins Wasserbad zu den anderen Kappen.

Jetzt beginnt die eigentliche Reinigungsarbeit. Wie Sie sehen, hat sich unter den Tastenkappen eine mehrere Millimeter dicke Schicht aus Staub, Haaren, Krümeln, Fingernägeln, Zähnen und so weiter angesammelt (kein Witz! Sie werden erstaunt sein...). Diese gilt es zu entfernen, damit sich die Tasten wieder leicht drücken lassen. Besorgen Sie sich einen Staubsauger und saugen Sie den verschmutzen Bereich gründlich ab. Möglicherweise löst sich der Schmutz nur, wenn Sie beispielsweise mit dem Schraubenzieher daran kratzen. Nachdem die gröbsten Verdickungen entfernt sind, sollten Sie noch mit einem feuchten (nicht nassen!) Lappen darüberwischen. Ersatzweise kann es auch sinnvoll sein, mit einem schnell verdunstenden Desinfektionsmittel über das Tastaturgehäuse zu sprühren, hier hat sich zum Beispiel »Softasept« (B.Braun AG, 3508 Melsungen, Best. Nr. 388514/3) recht gut bewährt. Achten Sie jedenfalls darauf, daß die Tastatur nicht zu feucht wird, damit später keine Kurzschlüssen entstehen.

Nachdem alles wieder schön saubär ist, setzen Sie alle Tastenkappen wieder auf. Wenn Sie sich nicht mehr daran erinnern können, wo welche Taste saß, sehen Sie sich ein Foto von einem C 64 oder auch die heimische Schreibmaschine (Z und Y vertauscht!) an. Sie können das Keyboard auch provisorisch wieder anschließen, dann vorsichtig den C 64 aktivieren und probieren, welche Taste welches Zeichen auf den Bildschirm bringt. Sollte die eine oder andere Kappe nicht stecken bleiben, versuchen Sie es nochmal, indem Sie die Kappe leicht schräg aufstecken. Keine Angst vor Verwechslung der beiden SHIFT-Tasten, diese sind absolut identisch. Bevorzugen Sie eine deutsche Tastatur, können Sie bei dieser Gelegenheit übrigens die Kappen <Z> und <Y> absichtlich vertauschen und später softwareseitig die Belegung umdefinieren.

Diese Generalreinigung muß normalerweise nur alle zwei bis drei Jahre

durchgeführt werden. Wichtiger ist es, die eigentlichen Tastenkontakte zu putzen, die sich im Innern des Tastaturgehäuses verbergen. Dieser Vorgang kann je nach Auslastung des Rechners schon bis zu vier- oder fünfmal im Jahr notwendig sein. Lösen Sie dazu auf der Rückseite der abgenommenen Tastatur (mit aufgesteckten Tastenkappen) die 23 kleinen Kreuzschlitzschrauben, so daß Sie die Tastaturplatine abnehmen können. Vorher müssen Sie noch die beiden Drahstücke ablöten, mit denen die SHIFT-LOCK-Taste angeschlossen ist. Bauen Sie hier eine Steckverbindung ein, werden spätere Reinigungen erleichtert. Keine Angst, es befinden sich keine kleinen Teile unter der Platine, die herausfallen könnten (auch keine großen). Wie Sie sehen, wird bei Betätigung einer Taste ein kleines auf einer Feder gelagertes Stück Leitgummi gegen zwei blanke Stellen auf der Tastaturplatine gedrückt, verbindet diese und stellt so den Kontakt her. Viel billiger geht's nicht mehr. Aber was soll's. Die blanken Stellen auf der Platine sollten als erstes von Staub gereinigt werden. Dazu hat sich folgende Vorgehensweise bewährt: Tränken Sie ein absolut fusselfreies Tuch (z.B. aus Filz, nur zur Not genügt auch ein unbenutztes Tempo-Taschentuch) mit Wasser oder besser - weil es schneller verdunstet - mit Nitro-Verdünner (Farblöser, z.B. von Brillux, das erhalten Sie zum Beispiel in Hobby- oder Farbengeschäften). Mit dem Tuch wischen Sie vorsichtig über die gesamte Platine und entfernen Staubreste. Bei Verwendung des Verdünners (entspricht Tipp-Ex-Verdünner) arbeiten Sie bitte bei weit geöffnetem Fenster, da diese Chemikalie gesundheitsschädlich ist. Mit dem Tuch reinigen Sie dann auch die 65 Leitgummis einzeln.

Jetzt wird die Tastatur wieder wie neu arbeiten. Sie muß nur noch zusammengebaut werden. Legen Sie die Platine wieder auf die Tastatur, löten oder stecken die Verbindung zu <SHIFT LOCK> wieder an und verschrauben das Ganze mit den 23 kleinen Schrauben. Danach kann das Keyboard in den Computer eingebaut werden. Ist auch das geschehen, stecken Sie den Tastaturstecker wieder auf die Leiste ganz links oben auf der Computerplatine. Die 18 Adern müssen links aus dem Stecker herausragen, falsches Einstecken ist mechanisch jedoch ausgeschlossen. Danach verschrauben Sie auch den Brotkasten wieder und schließen ihn an Trafo und Monitor an. Starten Sie einen Probelauf: Sie werden sich wundern, wie sehr sich die Qualität der Tastatur durch diese einfachen Handgriffe verbessert hat!

Der Hyper-Reset

Einen Reset-Taster haben wohl die meisten C 64-Besitzer. Was noch fehlt, ist die Möglichkeit, auch aus reset-geschützten Programmen auszusteigen. Mit dem Hyper-Reset ist das kein Problem. Unsere Schaltung (Bild) besteht aus drei Bauteilen, Software ist nicht notwendig. Der alte Reset-Taster T2 (nicht unbedingt notwendig) wirkt wie gewohnt, ein Druck auf den neuen T1 bricht auch Reset-geschützte Programme ab. Außerdem wirkt er nicht auf die Floppy. Der Einbau dauert knapp eine Viertelstunde und ist auch für den Laien unproblematisch.

Hinweis: Aus technischen Gründen kann es vorkommen, daß nach einem Kaltstart mit dem neuen Reset-Taster die Speichertestroutine falsch arbeitet. Sollte nach dem Reset eine andere Anzahl als 38911 BYTES FREE erscheinen, geben Sie

bitte folgende Befehle ein:

```
POKE 32774,0
SYS 64738
```

Der POKE-Befehl macht den Reset-Schutz unwirksam gegen den nun durch den folgenden SYS-Befehl ausgelösten Reset-Befehl. Dieses gelingt allerdings nur bei im Speicher befindlichen Programmen, nicht bei Steckmodulen am Expansionport.

Schmalschrift

Erweitern Sie Ihren MPS801-kompatiblen Drucker um eine Schmalschrift-Funktion, die bis zu 120 Zeichen pro Zeile druckt. Unser Programm bietet darüberhinaus noch einige Sonderfunktionen. Sie werden Ihren Printer nicht wiedererkennen!

Wenn Sie öfters mit Ihrem Drucker arbeiten, haben Sie sicher schon einmal den Wunsch gehabt, eine Passage schmäler als den übrigen Text auszudrucken. Vielleicht wollen Sie eine Tabelle oder Übersicht drucken, die 120 Zeichen breit ist, oder mehr Informationen auf einer Seite unterbringen, oder einfach Ihrem Schriftbild zu einem neuen Aussehen verhelfen. »Condensed« ermöglicht es jetzt endlich auch auf MPS 801 kompatiblen Druckern (MPS 803, VC 1525 und so weiter), die diese Option sonst nicht eingebaut haben, schmal zu drucken. Als Sonderfunktionen sind nicht nur die inverse Schrift, sondern auch Breitschrift und gesperrte Schrift verfügbar.

Laden Sie das Maschinenprogramm mit

```
LOAD "CONDENSED",8,8
```

und geben danach

```
NEW
```

ein. Jetzt kann mit

```
SYS 49152
```

die Schmalschrift am Drucker installiert werden. Sie ist bis zu einem Reset verfügbar. Das Programm druckt schlanke Versionen der Zeichen mit den Ascii-Codes 32 bis 95:

```
!"#$%&'()*+,-./0123456789:;<=>?ABCDEFGHIJKLMNOPQRSTUVWXYZ[£]^
```

sowie das Leerzeichen, den Klammeraffen und den Pfeil nach links. Die Grafikzeichen über 95 werden ganz normal gedruckt, Zeichen unter 32 werden wie gewohnt behandelt. Um die Schmalschrift einzuschalten, müssen Sie ein CHR$(1) zum Drucker schicken, CHR$(2) schaltet sie wieder aus. Dies wird so

verwendet, wie Sie es schon vom Ein- und Ausschalten der Breitschrift mit CHR$(14) und CHR$(15) bei diesen Druckern kennen. Also zum Beispiel so:

```
OPEN1,4:PRINT#1,"NORMALE..."+CHR$(1)+"UND SCHMALSCHRIFT"+CHR$(2)
```

Die Schmalschrift bleibt ggf. auch nach dem Zeilenende aktiv. Condensed benutzt darüberhinaus die gewohnten Druckercodes, um die reverse und doppelt breite Schrift ein- und auszuschalten:

```
CHR$(1)    schmal ein
CHR$(2)    schmal aus
CHR$(18)   revers ein
CHR$(146)  revers aus
CHR$(14)   breit ein
CHR$(15)   breit aus
```

Ist die doppelt breite Schrift im Schmal-Modus aktiviert, passen nur noch 60 Zeichen in jede Druckerzeile, also weniger als im normalen Textmodus des Druckers. Darüberhinaus können Sie auch gesperrt drucken. Dann wird der Zeichenabstand leicht erhöht. So lassen sich einzelne Abschnitte hervorheben.

```
CHR$(3)    gesperrt ein
CHR$(4)    gesperrt aus
```

Allerdings funktioniert die gesperrte Schrift nur im Schmalschriftmodus. Ansonsten lassen sich die Modi beliebig kombinieren. Die Schmalschrift hat nur bei der Ausgabe auf das Gerät Nr. 4 (Drucker) Wirkung, am Bildschirm z.B. läuft alles ganz normal. Nun wollen wir ein Programmlisting in Schmalschrift ausgeben:

```
OPEN1,4:CMD1,CHR$(1):LIST
```

gefolgt von

```
PRINT#1,CHR$(2):CLOSE1
```

Sie können natürlich auch z.B. ein gesperrt gedrucktes Listing ausgeben lassen. Dazu hängen Sie an CHR$(1) einfach noch +CHR$(3) an. Ebenso funktioniert es mit der Umschaltung auf reverse oder doppelt breite Schrift, oder eben mit Kombinationen.

Schmale und normale Schrift läßt sich auch innerhalb einer Druckzeile beliebig umschalten. Etwa gibt folgende Zeile den gesamten Zeichensatz der Schmalschrift aus:

```
OPEN1,4:FORI=32TO95:PRINT#1,CHR$(1)CHR$(I)CHR$(2)" = "CHR$(I):NEXT
```

Die CHR$-Codes lassen sich auch einfach innerhalb von Anführungszeichen mit CTRL-Kombinationen erreichen:

```
CHR$(1)  = <CTRL A>
CHR$(2)  = <CTRL B>
CHR$(3)  = <CTRL C> oder <RUN STOP>
CHR$(4)  = <CTRL D>
CHR$(18) = <CTRL R> oder <CTRL 9>
CHR$(146)= <CTRL 0>
CHR$(14) = <CTRL N>
```

CHR$(15) = <CTRL O>

Intern funktioniert Condensed wie folgt: Nach dem Start mit SYS 49152 wird der CHROUT-Vektor auf eine neue Routine verbogen. Jedesmal, wenn nun ein Zeichen ausgegeben wird, übergibt der C64 die Kontrolle an die Erweiterung. Diese prüft zunächst, ob die Ausgabe auf den Drucker erfolgt. Wenn nicht, wird an die gewöhnliche Ausgaberoutine übergeben. Ansonsten geht es in der Condensed-Routine weiter. Ist die Schmalschrift aktiviert, prüft das Programm, ob der Zeichencode zwischen 32 und 95 liegt. Falls ja, wird das Zeichen im Grafikmodus des Druckers in einer Matrix von 4x7 (normale Zeichen: 6x7) aus einer im Programm eingebauten Zeichensatztabelle ausgegeben. Zeichen, die außerhalb des erlaubten Bereiches liegen, werden normal ausgegeben. Falls es sich um Steuerzeichen handelt, verändert das Programm ggf. die Flags für revers, breit, schmal oder gesperrt.

Ein Beispiel zur Anwendung gibt das Demoprogramm. Es gibt eine ziemlich große Tabelle, die aus einem Kalkulationsprogramm wie »Tabula Rasa« stammen könnte, schmal aus. Dabei kommt übrigens auch die an anderer Stelle im Buch enthaltene Using-Routine zur Anwendung.

Liebesgrüße von Diskette: »Message-Maker«

Mit Hilfe des Programmes »Message-Maker« verfassen Sie einfach ansprechende Computerbriefe. Mit dem Editor wird der Text, der später erscheinen soll, geschrieben. Dabei werden auch alle Cursorbewegungen, Verbesserungen, Farbwechsel usw. gespeichert und später angezeigt. Verschiedene Wiedergabe-Geschwindkeiten sind einstellbar, Zusatzfunktionen wie Pause, auf Taste warten, Cursor-Position speichern, Wiederholen von beliebig langen Textteilen, Farbwechsel (auch Rahmen- und Hintergrundfarbe) sowie die Verwendbarkeit beliebiger Zeichensätze und - auf Wunsch - Musikstücke im Hintergrund machen den dennoch einfach zu bedienenden »Message-Maker« interessant.

Vor der Arbeit

mit dem Message-Maker müssen Sie sich eine eigene Message-Maker-Diskette zusammenstellen. Diese wird »Systemdisk« genannt und beinhaltet das Hauptprogramm »MESSAGE-MAKER«, das Systemfile »MESS.KOPF«, einen beliebigen Zeichensatz, der den unten angegebenen Bedingungen entsprechen muß unter dem Namen »MESS.FONT« und, bei Bedarf, ein Musikstück. Verwenden Sie dazu ein beliebiges Datei-Kopierprogramm und kopieren sich die genannten Files von der Diskette zum Buch.

Das Zeichensatzfile

muß den Namen »MESS.FONT« haben, neun Blocks lang sein und ab Adresse 12288 ($3000) beginnen. Die Zeichen sind im üblichen C64-Format gespeichert. Da ohne dieses File ein vernünftiges Arbeiten mit dem Message-Maker nicht möglich ist, können leider nicht die eingebauten C64-Systemzeichensätze verwendet werden. Dazu müßten Sie diese Fonts aus dem Char-ROM auslesen und auf Disk als »MESS.FONT« abspeichern. Nicht vergessen, die Startadresse

(z.B. mit einem Diskmonitor oder dem Programm »Chartranposer«) auf $3000 zu ändern! Eine Diskette mit 20 geeigneten Zeichensätzen ist gegen eine geringe Gebühr beim Autor erhältlich.

Falls Sie verschiedene Zeichensätze aus einer Bibliothek verwenden möchten, müssen Sie den gewünschten Zeichensatz mit dem Rename-Kommando der Floppy in »MESS.FONT« umbenennen.

Das Musikstück

das während der Wiedergabe des Textes im Hintergrund ertönen soll, ist optional. Wollen Sie es verwenden, speichern Sie es unter dem Namen »MESS.MUSIC« nicht auf die Systemdiskette, sondern auf die Diskette, auf der die fertige Message gespeichert wird. Dieses File muß nach dem absoluten Laden mit SYS 49152 gestartet werden können und soll dann die Musik im IRQ abspielen. Die nicht gecrunchten Files, die mit dem »Soundmonitor« aus der 64'er erzeugt werden, erfüllen diese Bedingungen.

Bedienung des Message-Makers

Legen Sie die Systemdiskette ein. Laden Sie das Programm mit dem Befehl

LOAD "MESSAGE-MAKER",8

und starten es mit

RUN

Die Benutzerführung erfolgt in Englisch. Die Aufforderung, die Systemdisk einzulegen, bestätigen Sie mit der Leertaste. Es werden nun die Files »MESS.KOPF« sowie »MESS.FONT« nachgeladen. Wenn Sie danach den Message-Maker unterbrechen, ist der Neustart nicht mehr mit RUN, sondern nur noch mit

POKE 49152,76 : SYS 49152

möglich. Siehe dazu auch den Abschnitt »erste Hilfe« weiter unten.

Hauptmenü

Es stehen drei Funktionen zur Verfügung, die mit den Funktionstasten abgewählt werden:

<f1> write: Text eingeben, alter Text wird gelöscht
<f3> quit: Programm beenden (Reset, Neustart wie oben beschrieben)
<f5> help: Anzeige aller Sondertastenfunktionen im Editor. Diese Tafel kann später von Editor aus nicht aufgerufen werden!

Der Editor

Die Eingabefunktion des Textes (Editor) wird bedient wie der normale Basic-Editor. Beachten Sie, daß alles, was Sie hier eingeben, später in eben dieser Form wiedergegeben wird. Schreiben Sie also ein Wort und löschen es mit der Taste <DEL> oder <f1> und <D> gleich wieder, so erscheint es bei der Wiedergabe, um danach gleich wieder gelöscht zu werden. Ebenso werden alle Cursorbewegungen sowie die Editorfunktionen Bildschirm löschen, Cursor home, revers on/off, Cursorfarbe ändern gespeichert.

Sollte nach dem Aufruf des Editors (<f1> im Hauptmenü) nur grafischer Unsinn

anstelle von Buchstaben auf dem Bildschirm erscheinen, wenn Sie eine Taste drücken, konnte das File »MESS.FONT« nicht geladen werden. Kontrollieren Sie, ob es sich auf der Systemdiskette befindet, einen vernünftigen Zeichensatz enthält und die übrigen oben genannten Bedingungen erfüllt.

Der Kommando-Modus

Drücken Sie im Editor die Taste <f1>, ertönt ein Pfeifton. Der Befehls- oder Kommando-Modus ist aktiv. Wenn Sie jetzt eine der folgenden Tasten drücken, wird nicht das entsprechende Zeichen ausgegeben, sondern ein Kommando aufgeführt.

<RUN STOP>: Beenden des Kommando-Modus, Rückkehr in den Editor
<K>: Key-Wait; später bei der Wiedergabe wird an dieser Stelle auf einen Tastendruck gewartet
<P>: Pause; später bei der Wiedergabe wird hier eine Pause eingelegt
<D>: Del 10; die letzten zehn Zeichen werden gelöscht (als ob Sie zehnmal die <DEL>-Taste drücken)
<S>: Show; der bisher eingegebene Text wird in der Form ausgegeben, in der er später in der Message erscheinen wird. Nach dem Textende gelangen Sie wieder in den Editor
<A>: Abort; hiermit können Sie den Editor verlassen. Eine Sicherheitsabfrage erscheint rechts oben (»SURE?«). Drücken Sie auf die Taste <J> oder <Y>, erscheint das Hauptmenü wieder, der Text geht verloren (siehe unten »Erste Hilfe«). Haben Sie den Text noch nicht gespeichert, geben Sie stattdessen <N> ein, um wieder in den Editor zu gelangen
<1>: Frame; die Farbe des Bildschirmrahmens wird verändert. Diese Änderung wird auch bei der Wiedergabe an dieser Stelle im Text durchgeführt und kann beliebig oft wiederholt werden
<2>: Background; die Farbe des Texthintergrundes wird verändert. Diese Änderung wird auch bei der Wiedergabe an dieser Stelle im Text durchgeführt und kann beliebig oft wiederholt werden
<V>: Velocity; rechts oben erscheint die Meldung »SPEED« und die aktuelle Geschwindigkeit. Stellen Sie jetzt mit dem Cursortasten eine neue Wiedergabe-Geschwindigkeit ein (255 = sehr schnell, 1 = sehr langsam, 206 = Wert nach dem Start der Wiedergabe) und drücken <RETURN>. Diese Änderung hat keinen Einfluß auf den Editor, sie wird bei der Wiedergabe ab dieser Stelle im Text ausgeführt und kann beliebig oft wiederholt werden. Eine wichtige Textpassage kann so zum Beispiel nicht nur durch eine besondere Farbe, sondern auch durch verminderte Geschwindigkeit kenntlich gemacht werden. Umgekehrt ist es möglich, komplizierte Bildschirmmasken, etwa später auszufüllende Formulare oder Blockgrafiken, in hoher Geschwindigkeit aufzubauen, um so die Geduld des Lesers nicht unnötig zu strapazieren.
<=>: store/recall Cursor; drücken Sie im Kommando-Modus die Taste <=> das erste Mal, wird die aktuelle Position des Cursors auf dem Bildschirm gespeichert und der Editor wieder gestartet. Drücken Sie danach im Kommando-Modus die Taste <=> erneut, wird der Cursor an die zuvor gemerkte Stelle gesetzt. Dies kann beliebig oft wiederholt werden; beim nächsten Druck auf <=> im Kommando-Modus wird wieder gespeichert.
<*>: set start/end repeat; mit dieser Funktion können Sie bei der Wiedergabe beliebig lange Textteile beliebig oft wiederholen lassen. Dazu drücken Sie am Anfang der Passage die Taste <f1> (Kommando-Modus) und die Taste <*>. Damit ist der Anfang festgelegt. Jetzt geben Sie den zu wiederholenden Text ein. Am Ende drücken Sie wieder im Kommando-Modus die Stern-Taste. Oben rechts erscheint »Series« und die Zahl 002. Wählen Sie jetzt mit den Cursortasten, wie oft die Passage wiederholt werden soll (zwischen 2 und 255 Mal). Drücken Sie dann <RETURN>. Die markierte Textpassage wird nicht im Editor, sondern nur bei der Wiedergabe so oft wiederholt, wie Sie es

vorgegeben haben. Diese Funktion kann beliebig oft wieder angewendet werden, Verschachtelungen sind nicht möglich.
<Pfeil nach links>: Save; wenn Sie mit dem Schreiben des Textes fertig sind, sollten Sie ihn als Message speichern. Dazu drücken Sie im Kommando-Modus die Taste <Pfeil nach links>. Jetzt erscheint das Untermenü

Text speichern

Hier haben Sie die Wahl zwischen:
<f1> save to disk und
<f3> show text, return to editor

Wenn Sie diese Funktion nur versehentlich aufgerufen haben, drücken Sie jetzt <f3>. Der komplette Text wird angezeigt und der Editor wieder gestartet. Drücken Sie <f1>, um den Text (die Message) zu speichern, erscheint die Frage, was bei der Wiedergabe nach dem Textende passieren soll:

<f1> reset
<f3> restart
<f5> ready

Drücken Sie <f1>, wird nach der Anzeige der Message ein Reset ausgelöst. Betätigen Sie <f3>, startet die Message am Ende von vorn (endlos). Wollen Sie, daß der Computer nach der Anzeige der Message in den Basic-Modus zurückkehrt (Anzeige von READY.), betätigen Sie <f5>.

Jetzt wird die Message im Speicher stark verkürzt, indem wiederkehrende Zeichen zusammengefaßt werden. Dies hat keinen sichtbaren Einfluß auf das spätere Aussehen des Textes, spart aber u.U. einigen Speicherplatz auf Diskette. Der Computer zeigt an, auf wieviel Prozent die Textlänge verkürzt werden konnte.

Legen Sie nun die formatierte Diskette ein, auf der die fertige Message gespeichert werden soll. Geben Sie sodann auf die Frage nach dem Namen, unter dem die Message auf Diskette gespeichert werden soll, einem maximal 16 Zeichen umfassenden Namen ein und drücken <RETURN>. Typische Namen sind »READ-ME«, »LIES MICH«, »HALLO«, »BITTE LESEN«, »INFO«, »MESSAGE«, »BRIEF« und so weiter.

Daraufhin wird die Message auf der Diskette gespeichert. Danach erscheint wieder das Hauptmenü des Message-Makers. Es ist nicht möglich, vom Message-Maker aus einen gespeicherten Text in den Editor zu laden und dort nachzubearbeiten.

Das Message-File

kann wie ein normales Basic-Programm mit dem Befehl LOAD "NAME",8 geladen und mit RUN gestartet werden. Es erscheint nun ein kleiner Hinweis des Message-Makers, der sinnvollerweise zum Abspielen des Disketten-Briefes nicht benötigt wird. Jetzt wird das Zeichensatz-File »MESS.FONT« geladen. Achten Sie also darauf, daß sich auch auf dieser Diskette dieses File befindet, und daß es den oben genannten Bedingungen entspricht. Falls ein Musikstück unter dem Namen »MESS.MUSIC« gespeichert ist, wird auch dieses automatisch geladen und mit SYS 49152 gestartet, sonst ertönt keine Musik.

Nach dem Laden dieser Files wird die Message angezeigt. Am Textende erfolgt, je nachdem, was Sie eingestellt haben, ein Reset, ein Neustart oder ein

Sprung in den Basic-Interpreter.

Erste Hilfe
Gültig nur für Version 1.0 des Message-Makers !

Falls der Editor versehentlich verlassen wurde (z.B. <f1>, <A>) oder ein Reset ausgelöst wurde, kann der Text, falls noch kein neuer eingegeben wurde, auf folgende Weise wieder zurückgeholt werden:

Ggf. Reset-Taster drücken. Um das Kopfprogramm wieder zu holen, von der Systemdiskette das Programm »MESS.KOPF« laden. Textendezeiger (166/167) auf das Byte hinter dem letzten Zeichen des Textes setzen. Dieses kann per Monitor (z.B. SM-Kit) herausgefunden werden, der Text beginnt bei 16384 = $4000). 166 enthält das Lowbyte, 167 das Highbyte der Adresse. Mit POKE 53272,31 den Zeichensatz wieder einschalten. Editor mit SYS 50490 wieder starten (nur Version 1.0 des Message-Makers). Zur Richtigstellung aller Zeiger den Text ansehen (<f1>, <S>). Während der Text gezeigt wird, prüfen, ob es noch Fehler gibt. Wenn ja, warten bis Textende (!), dann mit <RESET> aussteigen, den Fehler berichtigen (Monitor) und wie oben gezeigt fortfahren. Wenn alles in Ordnung ist, weiterschreiben.

Ändern der Standard-Wiedergabe-Geschwindigkeit im Kopffile (Head Version 1.1)

POKE 2232,X (normal = 50, sehr schnell = 1, sehr langsam = 255)

Hätten Sie's gewußt?

Die Trickkiste

Interessante Tips und Kniffe, die nicht im Handbuch stehen

Alle diejenigen, die bisher der Meinung waren, ihren Computer wirklich bis aufs letzte Bit zu kennen, sollten aufmerken. Hier stellen wir Ihnen einige Kniffe und Tips vor, die es wirklich »faustdick hinter den Ohren haben«. Wir verraten Besonderheiten der Basic-Befehle, die nicht im Handbuch stehen. Das eine oder andere »Aha-Erlebnis« ist sicher auch für Sie dabei! Weiter hinten kommen dann noch »herkömmliche« Tips und Tricks, die jeder Einsteiger nützlich findet.

Alle Hinweise gelten, soweit nicht anders angegeben, für alle Acht-Bit Computer von Commodore, also neben C 64 und C 128 auch für VC 20, PLUS/4, C 16, C 116, PET 2001 und die alten Geräte der CBM-Reihe.

Strings PEEKen

Es stimmt wirklich. Dem Commodore 64 ist es egal, ob eine Zahl oder ein String zwischen die Klammern der PEEK-Funktion gesetzt wird. Sie können ohne

weiteres PRINT PEEK(A$) schreiben, ohne eine Fehlermeldung zu erhalten. Auch PRINT POS(A$) besitzt diese syntaktische Immunität.

Der Wert, der zurückgegeben wird, hängt von der letzten numerischen Operation ab. Beispielsweise wirkt X=53280:PRINT PEEK(A$) wie PRINT PEEK(53280), ermittelt also die Farbe des Bildschirmrahmens.

Sie können auch ein Literal zwischen die Klammern setzen, so wie bei PRINT PEEK("HALLO"), allerdings ergibt dieser Ausdruck immer den Wert der Speicherzelle 0. Wenn Sie einen PEEK("STRING") dreimal in einer Zeile ausführen, beschwert sich das System dann aber doch mit einem ?OUT OF MEMORY ERROR. Aber was soll man auch anderes erwarten, wenn man Buchstaben dort verwendet, wo doch eigentlich Zahlen hingehören.

Die erste Dimension

Im Commodore-Basic sind bei dimensionierten Variablen elf Elemente Standard. Sie können ja zum Beispiel den Befehl PRINT A(10) oder A(10) = 19 verwenden, ohne vorher das Array A dimensioniert zu haben, und erhalten trotzdem keinen ?BAD SUBSCRIPT ERROR. Das funktioniert, weil Basic beim Start alle Arrays, egal ob String oder numerisch, mit dem Dimensionswert 10 (die Zählung beginnt bei Null, also elf Elemente) vorbelegt.

Das Kuriose dabei: Nachdem einer der beiden obigen Befehle abgearbeitet wuree, bewirkt DIM A(10) einen ?REDIM'D ARRAY ERROR, obwohl Sie doch noch gar kein Array dimensioniert haben!

Der nächste, bitte!

Der Befehl NEXT, der FOR..NEXT-Schleifen abschließt, wird gewöhnlich mit nur einer Variable versehen. Sie können ihn aber auch nutzen, um mehrere Schleifen gleichzeitig zu schließen. Die Folge NEXT A:NEXT B:NEXT C läßt sich einfach zusammenfassen zu NEXT A,B,C. Neben dem Zeitvorteil bei Eingabe und Ausführung des Programmes werden Programme so auch verkürzt.

Was ist CMD?

Vielleicht kennen einige von Ihnen schon den CMD-Befehl, der dazu dient, die Ausgabe auf ein neues Peripheriegerät, etwa den Drucker, umzuleiten. Klar, um ein Programmlisting auf dem Drucker auszugeben, schreiben Sie einfach: OPEN4,4:CMD4:LIST. Die Frage ist jetzt: Was bedeutet die Abkürzung CMD? Es ist ganz einfach: Die Bezeichnung steht für das englische Wort »CHANGE MAIN DEVICE«, also »Haupt-Gerät umschalten«.

Ein lästiger Nebeneffekt: Wird ein GET-Befehl ausgeführt, so werden alle bis dahin wirksamen CMD-Befehle wieder aufgehoben. Das kann, wenn man es nicht weiß, Ursache für unerwartete Programmierfehler sein.

Weiter, immer weiter

Der Befehl CONT kann nach einem Programmabbruch mit der RUN/STOP-Taste oder den Befehlen END oder STOP dazu verwendet werden, ein Programm an der Abbruchstelle fortzusetzen, wenn zwischendurch keine Änderungen am Programm durchgeführt oder Fehlermeldungen ausgegeben wurden. Was aber passiert, wenn CONT innerhalb eines Programmes zu finden ist? Ganz einfach: Dann wird CONT zur Endlosschleife. Übrigens unterscheiden sich die Befehle END und STOP wirklich nur dadurch, daß bei END die Meldung BREAK IN xxxx unterbleibt. Sonst sind diese Befehle identisch und austauschbar.

Verewigt in Silizium

Maler pflegen ihre Namen in die Ecke der Bilder zu schreiben, »Vandalen« sprühen ihre Namen auf Wände, ein jeder Simpel schreibt seinen Namen in jeden Winkel, Programmierer und Hardware-Designer verewigen sich im ROM. Wenn Sie einen C 128 besitzen, geben Sie doch einmal ein: SYS 32800,123,45,6 (leicht zu merken). Beim C 64 gibt es diese Art von Impressum leider nicht.

Wieviel ist ein Punkt wert?

Ja, wieviel denn? Nichts! Oder, besser gesagt: Null. Immer, wenn die Ziffer Null allein verwendet wird, kann man sie durch einen Punkt ersetzen. Basic führt Befehle mit dem Punkt sogar schneller aus als mit der Null. Anders gesagt,

```
10 POKE 53281,.:POKE 53281,1:GOTO 10
```

läuft schneller als die gleichbedeutende Zeile

```
10 POKE 53281,0:POKE 53281,1:GOTO 10
```

DATA-Tip

Nuller in DATA-Zeilen können auch einfach weggelassen werden. Das spart ein wenig Speicherplatz und Tipp-Arbeit. Statt

```
10 DATA 34,0,2,45,0,0,23,0,0,0,2
```

kann man auch schreiben:

```
10 DATA 34,,2,45,,,23,,,,2
```

Ebenso lassen sich die "" bei Leerstrings in DATAs einsparen. Bei Strings gibt es noch eine Besonderheit zu beachten: Sollen geSHIFTete Buchstaben in DATA Verwendung finden, müssen die Texte in Anführungszeichen gesetzt werden, da der Interpreter sie sonst in Tokens wandelt. Sonst können die Anführungszeichen wegfallen.

Inkompatible Zwillinge

Jeder Programmierer kennt den Befehl GOTO, der den Programmablauf an einer bestimmten Zeile fortsetzt. Wußten Sie, daß der Befehl GOTO bzw. GOSUB ohne eine Zahl dahinter so wirkt wie GOTO 0 bzw. GOSUB 0? Dies kann man nutzen, wenn es um jedes Byte geht, etwa bei Einzeilern. Außerdem kann GOTO in zwei separate Wörter getrennt werden: Also nicht GOTO 20, sondern GO TO 20. (Wenn Sie es nicht glauben, probieren Sie es aus! Auch wenn das im Englischen grammatikalisch eigentlich korrekter ist, wird GO TO so gut wie nie verwendet, und das aus gutem Grund. GO TO belegt nicht nur im Speicher zwei Bytes mehr (eines für das Leerzeichen, eines für das zusätzliche Token für TO), sondern ist auch gar nicht voll kompatibel zu seinem kompakten Bruder GOTO.

Beispielsweise ist gegen ON A GOTO 100, 200, 300 nichts einzuwenden, während ON A GO TO 100, 200, 300 nur einen ungläubigen ?SYNTAX ERROR provoziert. Mit dem GOTO, das ja direkt hinter IF verwendet werden darf, ist es das selbe: IF A=4 GOTO 100 macht seine Sache einwandfrei, IF A=4 GO TO 100 ist zum Scheitern verurteilt. Der Befehl GOSUB darf nicht zerlegt werden.

GOTO auf Trab gebracht

Generell gilt, daß der Aufruf von Unterprogrammen mit GOSUB schneller ist als mit GOTO. Häufig benutzte Unterprogramm gehören an den Programmanfang, müssen dann allerdings zuerst mit einem GOTO umgangen werden. Beispiel: Die folgende Routine beginnt ab Zeile 2, das Hauptprogramm kann wegen Zeile 1 dennoch mit RUN gestartet werden:

```
1 GOTO 10
2 PRINT "BITTE TASTE DRUECKEN!"
3 POKE 198,0:WAIT 198,1:POKE 198,0
4 RETURN
10 hier beginnt das Hauptprogramm
```

Malnehmen für Könner

Auch bei Multiplikationen läßt sich ein Geschwindigkeitsgewinn erzielen, indem die größere der beiden Zahlen, die malgenommen werden sollen, vor den Stern gestellt wird. PRINT 3463 * 2 ist also schneller als PRINT 2 * 3463, obwohl mathematisch gleichbedeutend (kommutativ). Nicht nur aus Geschwindigkeitsgründen sollten Sie die Potenzfunktion soweit als möglich vermeiden. Geben Sie einmal ein PRINT 7^2, Sie werden sich wundern! Besser ist PRINT 7*7.

Einer fehlt!

Die Funktion MID$() benötigt drei Parameter, um aus einem String einen bestimmten Teil herauszuschneiden, oder nicht? Nein! Es genügen auch zwei Parameter. Wird der dritte Wert weggelassen, ergibt MID$() einfach alle Zeichen beginnend bei dem, das durch den zweiten Parameter angegeben wird. Der Befehl PRINT MID$("TESTPROGRAMM",5) gibt »PROGRAMM« aus. Diese Kurzform ist dann nützlich, wenn Sie ein RIGHT$ ausführen möchten, aber nicht wissen, wie viele Zeichen der Endstring enthalten soll, nur, ab wo im Quelltext er beginnt.

Übrigens steht im Handbuch, daß beide Parameter bei MID$ von 0 bis 255 liegen dürfen. Das ist verkehrt. Der erste Parameter darf nicht Null sein, sonst erscheint ein ?ILLEGAL QUANTITY ERROR.

Professionell Nachladen

Gewöhnlich hat das Nachladen eines Maschinenprogrammes von einem Basicprogramm aus mit dem Befehl LOAD "CODE",8,1 einen lästigen Nebeneffekt: Das Basicprogramm wird von vorn gestartet. Diesen Effekt vermeiden Sie, indem Sie stattdessen schreiben: SYS 57812 ("CODE"),8,1:POKE 780,0:SYS 65493. Kurz eine Erklärung: Der erste SYS-Befehl setzt die File-Parameter, also den Namen, die Geräte- und Sekundäradresse. POKE 780,0 sagt dem System, daß geladen, kein VERIFY ausgeführt werden soll. Der nächste SYS-Befehl schließlich ruft die LOAD/VERIFY-Routine auf.

Dieser Trick ist nur für Besitzer eines C 64 interessant, auf dem C 128 bzw. C 16 steht ohnehin der BLOAD-Befehl zur Verfügung.

Joker

Mit Hilfe der »Joker« »*« und »?« läßt sich die Directory-Ausgabe genauer spezifizieren. Um beispielsweise ein Directory zu erhalten, in dem nur

PRG-Files enthalten sind, schreibt man einfach:

```
LOAD "$0:*=P",8
LIST
```

Sie können das P durch ein S, U oder R ersetzen, und erhalten dann alle SEQ, USR bzw. REL-Files.

Wenn Sie ein Programm laden wollen, aber nur den Anfang des Filenamens kennen, schreiben Sie

```
LOAD "FILENA*",8
```

Das wissen Sie schon. Wenn ein Filename mit dem Sternchen endet, erhält das DOS (Floppy-System) den Auftrag, File(s) zu bearbeiten, deren Name mit der Vorgabe vor dem Stern beginnt. Ebenso ist es möglich, unbekannte einzelne Zeichen im Namen durch das Fragezeichen zu ersetzen:

```
LOAD "FILENA?E",8
```

Aber wußten Sie schon, daß diese Spielereien auch beim Directory funktionieren? Beispielsweise wünschen Sie einen Ausdruck aller Files, die mit HAT beginnen und ein E als fünften Buchstaben haben:

```
LOAD "$:HAT?E*",8
LIST
```

Datenschutz

Jeder hat hin und wieder den Wunsch, ein Programm so auf Diskette abzuspeichern, daß nur er es wieder laden kann, sonst niemand. Hier sind zwei verschiedene Tricks, die es auch für den fortgeschrittenen Anwender schwer machen, ein Programm zu laden. Beide arbeiten auf allen 1541-kompatiblen Laufwerken.

Erstens, ein Basicprogramm läßt sich durchaus so speichern, daß es im Directory als SEQ oder gar als USR-File erscheint. Dazu hängen Sie nur den gewünschten Filetyp an den Filenamen an:

```
SAVE "BEISPIEL,S",8
```

oder

```
SAVE "BEISPIEL,U",8
```

Wenn Sie jetzt das Directory begutachten, werden Sie feststellen, daß aus Ihrem PRG-File im ersten Fall ein SEQ-File geworden ist, im zweiten Fall ein USR-File. Nun probieren Sie mal, das File mit

```
LOAD "BEISPIEL",8
```

wieder zu laden. Denkste! Ein ?FILE NOT FOUND ERROR erscheint, und die rote Floppy-Lampe blinkt. Das Programm kann nur auf die selbe Weise wieder geladen werden, in der es gespeichert wurde, also mit

```
LOAD "BEISPIEL,S",8
```

oder

```
LOAD "BEISPIEL,U",8
```

je nach Filetyp.

Noch gemeiner wird es, wenn Sie das Programm mit einem Nullbyte im Filenamen speichern:

```
SAVE CHR$(0)+"TESTNAME",8
```

Im Inhaltsverzeichnis erscheint der Name stark verstümmelt, zusammen mit einer völlig falschen File-Länge, so um die 10000 Blocks. Natürlich ist das File nicht wirklich so groß. Das Programm kann nun nur der laden, der den Kniff kennt:

```
LOAD CHR$(0)+"TESTNAME",8
```

Anders ist nichts zu machen!

Struktur ist alles!

Basic ist wirklich nicht die Sprache, für die sich Freunde von strukturierter Programmierung entscheiden - weit davon entfernt! Aber was bleibt dem Einsteiger auf dem C 64 anderes übrig als Basic? Dennoch soll hier ein Tip nicht fehlen, wie man Basicprogramme zumindest künstlich etwas besser strukturiert. Die folgenden Befehlszeilen sehen nicht besonders übersichtlich aus:

```
10 FORI=1TO20:FORJ=1TOI:A=4:GOSUB400:NEXTJ,I
```

Syntaktisch ist nichts gegen die Zeile einzuwenden. Wer Wert auf Struktur legt, schreibt stattdessen:

```
10 FOR I=1 TO 20
20 FOR J=1 TO I
30 A=4
40 GOSUB 400
50 NEXT J
60 NEXT I
```

Jedem Befehl wird eine eigene Zeile gegönnt. Jetzt wäre es schön, wenn man noch die Schleifen entsprechend einrücken könnte. Das Problem dabei ist nur, daß der Interpreter Leerzeichen am Anfang einer Zeile streicht. Dem helfen wir ab, indem wir einen Doppelpunkt an den Zeilenanfang stellen:

```
10 FOR I=1 TO 20
20 :   FOR J=1 TO I
30 :       A=4
40 :       GOSUB 400
50 :   NEXT J
60 NEXT I
```

Der Nachteil soll nicht verschwiegen werden: Dieser Programmteil wird nicht so schnell abgearbeitet, wie oben. Dafür ist er auch für fremde Programmierer einfach und übersichtlich zu durchschauen.

Vorsicht, Falle!

Eine Falle besonderer Art birgt der GET-Befehl, wenn er auf numerische Argumente angewandt wird. Etwa für eine Menüabfrage würde sich folgendes anbieten:

```
100 GET A
110 IF A=0 THEN 100
```

Ab Zeile 120 würde man dann je nach A in die einzelnen Menüpunkte verzweigen. Zeile 100 holt die gedrückte Taste nach A (es soll ja eine Zifferntaste gedrückt werden), in Zeile 110 wird diese Schleife weitergeführt, falls noch kein Tastendruck erfolgte.

Diese Konstruktion funktioniert, solange Sie nur Zifferntasten drücken. Sie bricht aber in sich zusammen, will sagen, ergibt einen ?SYNTAX ERROR, wenn ein unvorsichtiger Anwender stattdessen eine Buchstabentaste drückt. Sicher keine Auszeichnung für den Programmierer. Sie sollten GET daher grundsätzlich nur für Strings anwenden, die dann mit VAL in einen numerischen Wert gewandelt werden. Im Beispiel muß nur Zeile 100 geändert werden:

```
100 GET A$:A = VAL(A$)
```

Jetzt wartet das Programm auf Zifferntasten und steigt auch dann nicht aus, wenn Sie eine alphanumerische Taste drücken.

INPUT-Bug

Der INPUT-Befehl enthält leider einen Fehler. Die Prompts (Text in Anführungszeichen direkt nach INPUT) sollten nicht in die nächste Bildschirmzeile hineinreichen, da er sonst aufgrund eines Fehlers in den ROMs der älteren C 64-Modelle Teil der Eingabe wird.

Der Computer liest bei INPUT übrigens alles, was rechts vom Fragezeichen steht. Wenn sich also Grafik oder Text auf der selben Zeile rechts vom Fragezeichen befindet, wird dieser Bildschirminhalt zusammen mit den eingegebenen Daten gelesen und verursacht so ziemlich sicher Fehler. Stellen Sie vor INPUT also fest, daß der rechte Teil der Zeile gelöscht ist.

Der Code von Nichts

Basic-Programmierer verwenden die Funktion ASC() dazu, den ASCii-Code eines Strings zu ermitteln. Bei den Commodore-Rechnern hat ASC allerdings eine Schwäche: Sie ergibt einen ?ILLEGAL QUANTITY ERROR, wenn der String des Arguments leer ist. Verständlicherweise, denn ein Leerstring hat auch keinen Code. Dumm ist das nur, wenn beispielsweise ein File byte-weise gelesen und in ASCii-Codes zerlegt wird:

```
10 OPEN 2,8,2,"FILENAME,S,R"
20 GET#2,A$
30 A=ASC(A$)
```

Hier tritt in Zeile 30 eine Fehlermeldung auf, wenn in Zeile 20 ein Nullbyte gelesen wird. Der GET#-Befehl ergibt in diesem Fall nämlich einen Leerstring. Hier kann Abhilfe geschaffen werden, indem wir sicherstellen, daß die ASC-Funktion mit einem CHR$(0) gefüttert wird, wenn A$ leer ist. Dazu ändern wir Zeile 30 wie folgt:

```
30 A=ASC(A$+CHR$(0))
```

Ist A$ leer, ergibt die Addition mit CHR$(0) den gewünschten String: CHR$(0). Ist A$ nicht leer, stört die String-Addition nicht weiter, da ASC nur das erste Zeichen des Parameter-Strings berücksichtigt.

Alles relativ

Mit LOCATE positioniert man beim C 128 den Grafik-Cursor an eine Stelle des Hires-Bildschirms, DRAW zeichnet Punkte. Interessanterweise lassen sich beide Befehle auch relativ adressieren. Dazu wird ein Pluszeichen vor die Parameter gesetzt. LOCATE +20,+50 bewegt den Grafik-Cursor von seiner aktuellen Position um 20 Pixel nach rechts und 50 Pixel nach unten. DRAW 1,+40,+60 zeichnet gemessen von der aktuellen Position des Stiftes einen Punkt 40 Pixel weiter rechts, 60 Pixel weiter unten. Negative Werte führen zu Fehlermeldungen.

Alles Zufall?

Die Basic-Funktion RND liefert Pseudo-Zufallszahlen im Bereich zwischen 0 und 1. Setzt man N=RND(X), so sind die Werte N abhängig vom Argument X der Zufallsfunktion. Drei mögliche Erzeugungsarten sind vorgesehen:

X größer 0: Der genaue Wert des positiven Zahlenwertes X spielt keine Rolle, RND(2) und RND(1) ergeben die gleiche Zufallszahlenreihe, da hier ein fester Zahlenwert als »Keinzahl« (»Samen«) verwendet wird. Der neue Zufallswert wird nach einem einfachen Algorithmus aus dem alten gebildet. Der erste Samen wird nach dem Einschalten auf 0,811 635 157 gesetzt.

X = 0: Dieses Argument bewirkt, daß die Zufallszahlen abhängig vom Timer der CIA1 gebildet werden. Auch hier ist die erzeugte Zahlenreihe nicht wirklich zufällig.

X kleiner 0: Für negative Argumente ist die Zufallszahl eine Funktion des Arguments. Das heißt, daß hier bei jedem Aufruf eine neue Keimzahl gebildet wird. Bei der Verwendung der Systemzeit TI etwa wird stets in Abhängigkeit von TI ein neuer Samen gebildet: N=RND(-TI).

16 Funktionstasten abfragen

Mit einem genialen Trick lassen sich per Programm bis zu 16 Funktionstasten des C 64 abfragen. Wir kommen dabei ohne umständliche IF..THEN-Abfragen aus. Definieren Sie nur am Anfang Ihres Programms die Funktion

```
10 DEFFNA(X)=(X > 2)*(X < 7)*(((X-3-(X < 4)*4)*2)+(Y=0 OR Y=2))-(Y > 1)*8)
```

An entsprechender Stelle im Programm steht dann die Zeile

```
100 X=PEEK(197):Y=PEEK(653):IFFNA(X)=0 THEN 100
110 weiter im Programm
```

Wurde eine Funktionstaste gedrückt, macht der Computer mit Zeile 110 weiter. In X erhalten Sie nun bis zu 16 verschiedene Werte, die für folgende Kombinationen stehen:

(f 1), (f 3), (f 5), (f 7) allein
(f 2), (f 4), (f 6), (f 8) mit (SHIFT)
(f 9), (f11), (f13), (f15) mit (CBM)
(f10), (f12), (f14), (f16) mit (CTRL)

Das Eine oder das Andere, aber nicht beides

Gute Basic-Programmierer können durch die Verwendung der logischen Operatoren AND und OR ihre Programme verkürzen. Ein wichtiger Operator fehlt dem C 64 allerdings: Das ausschließende Oder (exclusive or, EOR). Bei AND ist das Ergebnis 1, wenn beide Operanden 1 sind. Bei OR erscheint die 1, wenn einer der beiden Operanden 1 ist. Was fehlt, ist die EOR-Verknüpfung, die eine 1 liefert, wenn genau ein Operand 1 ist, nicht aber beide.

EOR läßt sich aber mit AND und OR simulieren:

```
X = (A OR B) - (A AND B)
```

X ist das Ergebnis der bitweisen EOR-Verknüpfung von A und B.

Vergiß mein nicht!

Das Problem: In einem Basicprogramm soll der Wert einer Variable auch über den RUN-Befehl hinaus erhalten bleiben. Denken Sie an ein Basic-Spiel, bei dem der Highscore trotz Neustart mit RUN gespeichert werden muß. Die Lösung: Wir POKEn den Wert in eine unbenutzte Speicherzelle und holen ihn uns nach dem Start mit PEEK wieder. Das klappt bei ganzzahligen Werten von 0 bis 255. Soll ein Highscore in der Variable H (ganzzahlig, von 0 bis maximal 65535) gespeichert werden, kann er zunächst in High- und Lowbyte zerlegt werden:

```
200 HH=INT(H/256):HL=H-HH*256
```

Jetzt schreiben wir den Wert etwa in die unbenutzten Speicherzellen 701 und 702 und starten das Programm neu:

```
202 POKE 701,HL:POKE 702,HH:RUN
```

In der ersten Programmzeile lesen wir den Highscore aus:

```
10 H=PEEK(701)+256*PEEK(702)
```

Es tritt ein Problem auf: Nach dem Einschalten des Rechners stehen in 701 und 702 zufällige Werte, daher erhält die Variable HH bei ersten Start des Programms einen Zufallswert. Wir sollten daher dem Computer auch melden, daß sich gültige Werte in 701 und 702 befinden, etwa durch den speziellen Wert 123 in der Zelle 703, der dort nach dem Einschalten gewöhnlich nicht steht:

```
202 POKE 701,HL:POKE 702,HH:POKE 703,123:RUN
10 H=0 : IF PEEK(703)=123 THEN H=PEEK(701)+256*PEEK(702)
```

Sollte das Programm das erste Mal gestartet werden, erhält H den Wert 0, sonst den Wert vor dem Neustart in Zeile 202.

Gleichungen lösen

Das folgende kleine Programm löst Gleichungen!

```
10 DEF FN A (X)=
20 DEF FN B (X)=
30 A=1:S=1
40 FOR T=1 TO 9
50 FOR I=A TO 10^7 STEP S
60 IF FNA(I) > FNB(I) THEN 80
```

```
70 NEXT I
80 A=I-S: S=S*.1: NEXT T
90 PRINT"X =";X
```

In Zeile 10 und 20 müssen die linke und rechte Seite der Gleichung eingesetzt werden. Beispiel: Es soll die 27. Wurzel aus 5844 ermittelt werden, also X hoch 27 = 5844. Die Zeilen lauten:

```
10 DEF FN A (X) = X ^ 27
20 DEF FN B (X) = 5844
```

Nach dem Start mit RUN erscheint das richtige Ergebnis: X=1.37882068. Auch komplizierte Konstrukte wie (40+X)/40=40/X sind kein Problem:

```
10 DEF FN A (X) = (40+X)/40
20 DEF FN B (X) = 40/X

RUN

X=24.7213596
```

Hinweis: Die zu bearbeitenden Gleichungen dürfen keine quadratischen Gleichungen sein. Das Ergebnis muß positiv und kleiner als 10 Millionen (vgl. Zeile 50) sein. Ermittelt das Programm als Ergebnis -.1111111, so war die Aufgabe nicht lösbar.

Autostart

Wollten Sie schon einmal ein Basicprogramm schreiben, das sich nach dem Laden automatisch selbst startet? Hier ist die Methode, bei der Sie keinerlei Zusatzprogramm benötigen, ein »Kochrezept«. Sie brauchen weder Programmierkenntnisse in Basic oder gar Maschinensprache, noch ein Programm abzutippen. Fügen Sie lediglich zu dem Programm, das einen Autostart bekommen soll, in die allererste Zeile die beiden Befehle:

```
1 POKE 770,131:POKE 771,164
```

ein. Legen Sie jetzt eine formatierte Diskette mit genügend Platz ein und geben im Direktmodus in einer einzigen Zeile folgendes ein:

```
POKE 770,113:POKE 771,168:POKE 44,3:POKE 43,0:SAVE "FILENAME",8
```

Nach dem Speichern wird der Computere abstürzen, das ist normal. Schalten Sie das Gerät aus und wieder ein. Um nun den Autostarter zu laden, geben Sie ein:

```
LOAD "FILENAME",8,1
```

Wichtig ist der Zusatz ,1 nach dem normalen Ladebefehl. Stören Sie sich nicht daran, daß sich der Bildschirminhalt während des Ladens ändern wird, das Programm wird nach dem Laden automatisch starten. Wollen Sie nicht, daß man Zeichen auf dem Bildschirm sieht, während das Programm geladen wird, fügen Sie einfach vor die oben angegebenen vier POKEs den Befehl

```
PRINTCHR$(147):
```

ein. Dabei wird der gesamte Bereich ab Speicherzelle 768 gespeichert, Sie können dem Selbststarter also auch noch Nebenbedingungen mit auf den Weg

geben, die nach LOAD ...,8,1 automatisch aktiv werden: Zum Beispiel

```
POKE 808,239 (RUN STOP verriegeln)
POKE 792,193 (RESTORE-Taste verriegeln)
POKE 775,191 (LIST verriegeln)
```

Diese Befehle müssen vor dem speziellen Speichern wie oben zu sehen eingegeben werden.

Die Notbremse

Fast so etwas wie das Gegenteil des vorangegangenen Tips: Leider enthält der C 64 »ab Werk« keinen eingebauten Reset-Taster. Dies wäre ein Knopf, mit dem der Rechner in den Einschaltzustand versetzt werden kann, beispielsweise wenn ein Programm abgestürzt ist. Für uns ist das aber kein Problem, man kann nämlich die RESTORE-Taste (rechts über der RETURN-Taste) in ihrer Funktionsweise ziemlich frei umdefinieren, beispielsweise einen Reset-Schalter daraus machen. Die beiden Befehle

```
POKE 792,226:POKE 793,252
```

erledigen das für uns. Wenn Sie jetzt die RESTORE-Taste (auch ohne RUN/STOP) betätigen, wird ein Reset ausgelöst, der C 64 wird in einen definierten Zustand zurückversetzt, die Einschaltmeldung erscheint. So werden beispielsweise verschiedene Arbeitsspeicherzellen mit sinnvollen Werten versorgt, die während des Betriebes verändert wurden. Dabei wird übrigens auch die Umbelegung von RESTORE widerrufen! Sie könnten nun mit einer Renew-Routine Ihr verlorenes Basicprogramm wiederholen.

Renew

Wie oft passiert es, daß man versehentlich den Befehl NEW eingibt und sich gleich darauf auf die Finger schlagen möchte: Studenlange Programmierarbeit scheint rettungslos verloren, weil das Basicprogramm dummerweise nicht gespeichert wurde. Den gleichen Effekt hat ein vorschneller Druck auf den Reset-Taster (vgl. vorher): Das Programm ist weg.

Aber halt, es ist nicht ganz verschwunden. Eigentlich ist es noch im Speicher des C 64, aber versteckt. Wenn nach dem Löschen noch keine weiteren Programmzeilen eingegeben wurden, helfen folgende Befehle, das Programm zu retten:

```
POKE 2050,8
SYS 42291
POKE 46,PEEK(35)-(PEEK(781) größer 253)
POKE 45,PEEK(781) + 2 AND 255
CLR
```

Wohlgemerkt dürfen vorher keine Basic-Zeilen eingegeben oder Variablen definiert worden sein (fatal wäre z.B. A=56), da sonst das Programm rettungslos verloren geht.

Diese oder verwandte Befehlskombinationen findet man oft in Basic-Erweiterungen. Der entsprechende Befehl heißt dann OLD oder RENEW, da er den NEW-Befehl rückgängig macht.

Seitenweise IF..THEN

Irgendwann kommt Ihnen einmal ein Programm unter, in dem viele Zeilen zum Beispiel so aussehen:

```
60 IF A=5 THEN B=7
70 IF A=6 THEN B=13
80 IF A=7 THEN B=-3
90 IF A=8 THEN B=6
```

Je nach Wert von A soll also B einen Wert zugewiesen bekommen. Aber so umständlich? Sagen wir, A ist eine Integervariable zwischen 5 und 21. Sie würden nach obigem Verfahren 17 IF..THEN-Befehle brauchen, um alle Möglichkeiten abzudecken; fast einen Bildschirm voll. Diese Befehle brauchen viel Platz und viel Programm-Rechenzeit; beides ist rar und teuer auf dem C 64. Die 17 Zeilen könnten aber durch einen einfachen Dreizeiler ersetzt werden. Am Beginn Ihres Programms müßte folgendes stehen:

```
10 DIM ZZ(17):FOR X=1 TO 17:READ ZZ(X):NEXT
12 DATA 7,13,-3,6, und so weiter
```

Der DATA-Befehl wird mit den Werten von B abhängig von A gefüllt. Später im Programm erfüllt dann jedes Mal, wenn er gebraucht wird, ein einfacher Zuweiser wie

```
60 B=ZZ(A-4)
```

die ganze Arbeit für uns. Die Zeilen ab 70 fallen weg. Da die Zahl in A zwischen 5 und 21 liegt, subtrahieren wir 4, um in den Bereich des ZZ-Arrays (1 bis 17) zu gelangen. Die Technik spart Speicherplatz und bringt einen deutlich spürbaren Zeitgewinn.

Raus aus dem Quote-Modus

Der Anführungszeichen-Modus (quote-mode) des C 64 ist sowohl eine nützliche wie auch eine frustrierende Angelegenheit. Wenn Sie schon programmiert haben, wissen Sie, wie angenehm es ist, Bildschirmbefehle wie Bildschirm löschen, Farbe ändern, Cursor bewegen einfach in PRINT-Befehle einzubauen. Aber Sie wissen auch, in welche »Schwulitäten« Sie kommen, wenn Sie editieren wollen, während Sie sich im Quotemodus befinden (nämlich nach der Eingabe eines Anführungszeichens mit SHIFT 2): Der Computer führt Ihre Cursorbewegungen nicht mehr aus, sondern vermerkt sie in Form von inversen Steuerzeichen innerhalb des Textes. Allerdings gibt es einige Möglichkeiten, dieser Betriebsart ohne Umstände zu entwischen:

· Die RETURN-Taste schaltet grundsätzlich den Quote-Modus, den Einfüge-Modus und den Invers-Modus ab. Der Einfüge-Modus hat die gleichen Eigenschaften wie der Quote-Modus, allerdings werden hier auch Korrekturen mit (DEL) als Steuerzeichen (inverses T) dargestellt.
· Ein manchmal nicht erwünschter Nebeneffekt der RETURN-Taste, nämlich die Übernahme der eingegebenen Zeile in den Speicher, tritt bei Druck auf (SHIFT RETURN) nicht auf. Sie können danach den Cursor wieder nach oben bewegen und Korrekturen vornehmen.
· Beide Arten von (RETURN) bringen Sie in die nächste Bildschirmzeile. Um den Quote-Modus ohne »Platzverweis« zu stornieren, geben Sie einfach noch ein Anführungszeichen ein (SHIFT (2)) und drücken danach (DEL).

Diese Tricks arbeiten prima, wenn Sie ein Programm oder einen Text editieren, aber was tun, wenn man vom Programm aus einen zum Beispiel durch GET von Tastatur oder File eingeschalteten Anführungszeichen-Modus

abschalten will? Der Computer befindet sich ja immer dann in dieser Betriebsart, wenn er ein Anführungszeichen auf dem Bildschirm ausgegeben hat. Um sicherzustellen, daß der Modus abgeschaltet ist, geben Sie einfach einen POKE 212,0 (auf dem C 64, bzw. POKE 203,0 auf einem C 16).

MSE als Kopierprogramm

Die einfachsten Ideen sind oft die besten. Die Eingabehilfe des 64'er-Magazins, der MSE, läßt sich als Kopierprogramm verwenden. Dazu laden Sie den MSE und starten ihn ganz normal. Vom MSE laden Sie das zu kopierende Programm und speichern es dann mit der Tastenkombination (CTRL S) auf eine andere Diskette oder Kassette. Diese Technik funktioniert mit allen Maschinen- oder Basicprogrammen und Datenfiles, die im Directory mit PRG vermerkt sind, allerdings lassen sich die meisten kopiergeschützten Programme damit nicht kopieren.

Lange Zahlenkolonnen

Oft werden zum Beispiel mit Schleifen wie dieser Zahlenlisten auf dem Schirm ausgegeben:

```
100 FOR I=1 TO 1000: PRINT AR(I): NEXT
```

Hier soll der Inhalt des 1000 Felder umfassenden Feldes AR() ausgegeben werden. Eine lange Folge von 1000 dahingeschmissenen Zahlen ist die Folge, ein Mensch wird kaum mitlesen oder kontrollieren können. Die CTRL-Taste verlangsamt zwar leicht, aber nicht genug. Wenn es Ihnen zu schnell geht, bauen Sie einfach einen Befehl wie WAIT 198,1,1:POKE 198,0 ein. Oben wäre das also:

```
100 FOR I=1 TO 1000: PRINT AR(I): WAIT 198,1,1:POKE 198,0: NEXT
```

Die Nummern erscheinen ganz normal auf dem Bildschirm, allerdings nur so lange, bis Sie eine Taste drücken. Dann hält der Computer so lange an, bis Sie noch eine Taste drücken. Es klingt seltsam, aber der dritte Parameter beim relativ unbekannten Befehl WAIT 198,1,1 sorgt dafür, daß der Computer so lange wartet, bis der Inhalt der Speicherzelle 198 (= Anzahl der bisher gedrückten Tasten) gerade ist. Wenn Sie eine Taste drücken (eine ungerade Anzahl), wartet der Computer so lange, bis Sie eine weitere Taste drücken. Diese Technik findet Anwendung, wenn Sie zum Beispiel mit PEEK einen großen Speicherbereich oder wie hier eine dimensionierte Variable durchsehen.

Weitere nützliche WAIT-Befehle

Da wir gerade den WAIT-Befehl behandelt haben, nutzen wir die Gelegenheit und führen Ihnen weitere äußerst trickreiche Anwendungen dieses Mauerblümchens vor. Der Befehl hat die Syntax

```
WAIT Adresse, Maske1 (, Maske 2)
```

und wartet, bis der Inhalt der angegebenen Speicherzelle ggf. exklusiv-oder (vgl. oben) verknüpft mit der zweiten Maske (falls eine angegeben ist) und danach und-verknüpft mit der ersten Maske einen Wert ungleich Null ergibt. Da sich nur diese Speicherzellen selbständig ändern, wird WAIT fast nur im Zusammenhang mit Ein-/Ausgabe-Adressen verwendet.

Ein Beispiel: Beim Commodore 64 findet sich in Speicherzelle 653 die Information, welche der Taste(n) (SHIFT), (CBM) und/oder (CTRL) gedrückt

ist/sind. Bit 2 (Wertigkeit 4) dieser Zelle wird genau dann auf 1 gesetzt, wenn die CTRL-Taste gedrückt wird. Wollen Sie in Ihrem Programm darauf warten, daß der Anwender die CTRL-Taste drückt, geben Sie einfach den Befehl

WAIT 653,4

Man kann auch den Befehl geben, der Computer soll so lange warten, bis die CTRL-Taste (falls sie bei Erreichen des WAIT-Befehles denn gedrückt war) losgelassen wurde:

WAIT 653,4,4

ist die einfache Lösung. Für die SHIFT-Taste schreiben Sie statt der 4 eine 1 oder eine 2 für die Commodore-Taste. Mit Hilfe der SHIFT-LOCK-Taste können wir somit in jedes Basic-Spiel eine Pause-Funktion einbauen: Setzen Sie einfach in die Hauptschleife des Spieles (z.B. Bewegung der Spielfigur) den Befehl

WAIT 653,1,1

Das Spiel kann durch Einrasten der Taste SHIFT LOCK gestoppt werden, nach dem Entrasten geht es weiter.

Die Zelle 197 enthält einen speziellen Code der momentan gedrückten Taste oder die 64, wenn keine Taste gedrückt wird. Also können wir mit

WAIT 197,63

darauf warten, daß irgend eine Taste gedrückt wird, und mit

WAIT 197,64

darauf warten, daß alle Tasten losgelassen werden.

Wie bei allen Tricks dieser Rubrik spielt es auch hier keine Rolle, daß Sie die Kniffe verstehen, Sie sollten lediglich wissen, wie man sie anwendet, sozusagen das »Kochrezept«.

Die eingebaute Uhr des C 64 arbeitet mit den Speicherzellen 160 bis 162. Das machen wir uns zunutze, indem wir mit WAIT eine Pause von vorgegebener Länge erzeugen. Erst setzen wir die Uhr auf Null, dann warten wir, bis eine bestimmte Zeit vergangen ist, bis also ein bestimmter Wert im Uhrenregister steht. Die folgende Befehlsfolge erzeugt eine Zwangspause von 0,5 Sekunden:

POKE 162,0:WAIT 162,32

Die folgenden beiden gleichwertigen Zeilen erzeugen eine Wartezeit von 4-4/15 Sekunden:

POKE 161,0:POKE 162,0:WAIT 161,1

oder

TI$="000000":WAIT 161,1

Unverständliche Fehlermeldungen

Es gibt drei Fälle, da erscheinen nach der Eingabe fast aller Basicbefehle

Fehlermeldungen, die sich aber nicht erklären lassen, weil der Befehl völlig korrekt war.

· Im ersten Fall reagiert der C 64 auf jede Eingabe mit einem ?FORMULA TOO COMPLEX ERROR, Befehle werden überhaupt nicht mehr ausgeführt. Meistens ist die Ursache ein abgestürztes Programm oder ein fehlerhafter POKE, der den Computer scheinbar lahmlegt. Um diesen Effekt abzustellen, reicht ein

```
POKE 24,0
```

· Der zweite Fall: Der Rechner reagiert auf viele Eingaben nur noch störrisch mit ?SYNTAX ERROR. Das liegt häufig daran, daß eine falsche Zahl in Speicherzelle 2048 den Basic-Speicher »verschmiert«. Mit

```
POKE 2048,1
```

können Sie diesen unangenehmen Effekt gar selbst provozieren. Treiben Sie damit Freunde und Bekannte zum Wahnsinn, denn denen wird es jetzt nicht mehr gelingen, Programm zu editieren oder mit RUN zu starten. Im Regelfall wird diese Situation aber nicht künstlich herbeigeführt, sondern entsteht durch einen Unfall. Wie kann dann die volle Funktionsfähigkeit des Computers ohne Programmverlust wiederhergestellt werden? Durch einen einfachen

```
POKE 2048,0
```

klappt alles wieder wie gehabt.

· Drittens: Sie haben ein Maschinenprogramm absolut geladen, also mit dem Befehl LOAD "NAME",8,1. Solche Programme sind oft Hilfsprogramme und lassen sich beispielsweise mit SYS 49152 starten. Den Versuch, Variablen anzulegen oder Programme einzugeben oder zu starten quittiert der Computer allerdings gnadenlos mit einem ?OUT OF MEMORY ERROR, der einfach nicht verschwinden will. Abhilfe schafft ein einfacher NEW-Befehl, der zwar dem Maschinenprogramm nicht weh tut, aber das im Speicher stehende Basicprogramm löscht. Dieses holen Sie sich dann ggf. mit dem oben vorgestellten RENEW-Trick oder einem entsprechenden Hilfsprogramm zurück.

Schnelles Löschen von Zeilen

Leider fehlt dem Basic 2.0 des C 64 ein DELETE-Kommando, mit dem gezielt Zeilenbereiche eines Basicprogramms gelöscht werden können. Überflüssig zu erwähnen, daß das Löschen von vielleicht 40 oder 50 Zeilen sehr anstrengend und langweilig sein kann (Eingabe der ersten Zeilennummer, <RETURN>, Eingabe der zweiten Zeilennummer, <RETURN>, Eingabe der dritten Zeilennummer, <RETURN>, Eingabe der vierten Zeilennummer, <RETURN>, und immer so weiter). Zwar existieren Hilfsprogramme für diesen Zweck, aber wir wollen zeigen, wie es ganz einfach und effektiv geht.

Eine Lösung wäre ein Einzeiler, der die erforderlichen Nummern einfach auf den Schirm schreibt, wie

```
FOR I=3000 TO 3200 STEP 10:PRINT I:NEXT
```

Diese Zeile listet 20 Nummern im Bereich zwischen 3000 und 3200 (Schrittweite 20) auf dem Schirm. Um diesen Bereich zu löschen, tippen Sie einfach auf jeder Nummer <RETURN>. Wenn Ihr Programm nicht mit der Schrittweite 10 geschrieben wurde, fehlen vielleicht einige Zeilen, dann müssen Sie die Schrittweite STEP verändern. Glücklicherweise geht es noch

simpler: Geben Sie erst einmal POKE 774,0 ein. Dieser Befehl schaltet den LIST-Befehl so, daß er nur die Zeilennummern zeigt (ausprobieren!). Jetzt holen wir uns den gewünschten Bereich mit einem einfachen

```
LIST 3000 - 3200
```

auf den Schirm und löschen mit <RETURN> jede Zeile. Danach schalten wir mit POKE 774,26 wieder den Normalbetrieb von LIST ein (oder <RUN/STOP-RESTORE> drücken).

LIST in Basicprogrammen

Normalerweise ist es nicht möglich, den LIST-Befehl innerhalb von Basicprogrammen zu verwenden, da der C 64 die Bearbeitung des Programms nach dem LIST-Befehl stoppt. Mit einem kleinen Trick ist es aber trotzdem möglich. Das folgende Beispiel listet innerhalb des Programms die Zeile 10.

```
10 POKE768,61:SYS42980,LIST10:POKE768,138
20 WAIT198,1: POKE198,0: PRINT"WEITER": GOTO10
```

Natürlich können Sie in Zeile 10 anstelle des LIST 10 auch zum Beispiel LIST 10- oder LIST 400-500 usw. schreiben. Vergessen Sie nicht das Komma nach dem SYS-Befehl!

Vorgaben

Viele Programmierer verwenden einen Befehl wie den folgenden, wenn Sie vom Anwender eine Eingabe erwarten und ihm schon eine mögliche Antwort vorgeben wollen:

```
INPUT"SIND SIE SICHER(2 SPACES)J(3 LEFT)";A$
```

Hier wird die Frage gestellt: SIND SIE SICHER? zusammen mit einem »Default«. Ein Default ist eine Vorgabe, in diesem Fall die Antwort »J«, die dem Programm übergeben wird, wenn der Anwender nur <RETURN> drückt, ohne etwas einzugeben. Dieser Trick funktioniert meistens ohne weiteres, die Cursor-Steuer-Kommandos sorgen dafür, daß sowohl die Vorgabe »J« (Sie hätten ebenso gut »N« schreiben können) wie auch das Fragezeichen, das der INPUT-Befehl selbst erzeugt, auf dem Bildschirm an der richtigen Stelle stehen.

Sie erkennen das Problem: Was tun, wenn die Länge des Defaults vorher nicht bekannt ist? Wenn die Vorgabe in einer Variable steht? Dann behelfen Sie sich am besten so:

```
PRINT"PROMPT ";X;:POKE211,6:INPUTX
```

Hierbei meint »PROMPT« den Abfrage-Text, zum Beispiel »Sind Sie sicher«. Der Befehl POKE 211,6 bewirkt, daß der Cursor in die 6. Bildschirmspalte gesetzt wird. Sie müssen je nach Länge des Prompts diesen Wert variieren (hier »PROMPT« = 6 Zeichen, daher 6). Verwenden Sie diese Technik mit einer String-Variable anstelle von X, schreiben Sie zwei Leerzeichen nach dem Prompt anstelle des einen.

Kommas im INPUT-Befehl

Nochmal INPUT: Der INPUT-Befehl, mit dem der C 64 ausgerüstet ist, hat - neben anderen Schwächen - einen Nachteil: Man kann keine Kommas eingeben.

Ein Beispiel:

```
10 INPUT"TEXT";T$
20 PRINT"ES WAR";T$
```

Geben Sie hier in Zeile 10 einen Text ein, der Kommas oder Doppelpunkte enthält, so wird sich der Rechner mit einem ?EXTRA IGNORED beschweren und - wie beim folgenden PRINT-Befehl zu sehen ist - den Teil nach dem Komma bzw. Doppelpunkt gnadenlos abtrennen. Abhilfe: Ein Anführungszeichen, eingegeben vor dem Text. Dieses geht nicht in die Variable T$ ein, sorgt aber dafür, daß Sie in der Eingabe alle Zeichen verwenden dürfen (ausgenommen sind natürlich weitere Anführungszeichen).

Fehlerkanal im Direktmodus, Teil I

Die rote Fehler-LED der Floppy blinkt hektisch, ein Fehler ist aufgetreten. Aber welcher? Leider kann man normalerweise im Direktmodus nicht den Floppy-Fehlerkanal auslesen, das ist wegen dem dazu notwendigen INPUT#-Befehl nur innerhalb von Basic-Programmen möglich. Falls sich ein wichtiges Programm im Speicher befinden, das Sie nicht überschreiben wollen, stehen Sie jetzt ohne folgenden Trick ganz schön »auf dem Schlauch«.

Unter Zuhilfenahme einiger Maschinensprache-Systemroutinen des Computers (wie das funktioniert, müssen Sie nicht verstehen, Hauptsache, es klappt) gelingt es uns, auch im Direktmodus Dateien, zum Beispiel den Fehlerkanal zu lesen. Dazu öffnen wir erst den Kanal wie gewohnt mit

```
OPEN 15,8,15
```

Dann wird - im Direktmodus - folgende Zeile eingegeben:

```
FOR X=0TO40: POKE781,15: SYS65478: SYS65487: SYS65490: SYS65484: IF ST= 0
THEN NEXT
```

Es erscheint die Meldung, zum Beispiel

```
26, WRITE PROTECT ON,18,01
```

Danach wird (falls nicht mehr benötigt) der Kanal wieder geschlossen.

```
CLOSE 15
```

Dieser Trick klappt immer dann, wenn im Direktmodus eine Datei gelesen werden soll. Der Befehl SYS 65490 sorgt für die Ausgabe auf dem Bildschirm. Sie müssen ggf. nur nach dem POKE 781, die Filenummer des geöffneten Files (hier: 15) einsetzen. Beispielsweise listet folgendes »Programm«, das allerdings nur im Direktmodus läuft, eine Datei auf dem Drucker:

```
OPEN 1,8,2,"NAME DER DATEI,S,R"
OPEN 2,4
FOR X=1 TO 10000000: POKE 781,1: SYS 65478: CMD 2: SYS 65487: SYS 65490: SYS
65484: IF ST=0 THEN NEXT
CLOSE 1:CLOSE 2
```

Nach der Übertragung kann man mit

```
PRINT X
```

die Anzahl der gedruckten Zeichen erfragen.

Fehlerkanal im Direktmodus, Teil II

Haben Sie die obige Methode schon ausprobiert? Es geht auch einfacher: Wenn es dem Computer nicht »paßt«, daß wir im Direktmodus den INPUT#-Befehl anwenden, dann müssen wir ihm eben »vorgaukeln«, daß er sich im Programm-Modus befindet. In den Speicherzellen 57 und 58 steht während des Ablaufs eines Basicprogramms die Zeilennummer der gerade bearbeiteten Basic-Zeile, im Direktmodus ein spezieller Code. Schreiben wir direkt vor dem INPUT#-Befehl eine Null in diese Zellen, dann denkt der Computer, er befindet sich im Programm. Er meckert dann nicht bei der Abfrage des Fehlerkanals:

```
POKE 57,0:POKE 58,0:OPEN 1,8,15:INPUT #1,A,B$,C,D:PRINTA;B$;C;D: CLOSE 1
```

Die beiden POKEs haben keine Nebenwirkungen, sie werden automatisch sofort am Zeilenende bei CLOSE 1 wieder korrigiert.

Directory ohne Programmverlust

Häufig möchte man das Inhaltsverzeichnis einer Diskette in den Speicher bringen und listen, ohne das dabei im Speicher befindliche Programm zu löschen. Haben Sie nicht gerade eine Erweiterung wie DOS 5.1 zur Hand, hilft folgender Kniff:

```
POKE 44,PEEK(46)+1
```

Der Basic-Speicher wird auf einen freien Speicherplatz umgestellt. Jetzt können Sie wie gewohnt das Directory mit LOAD "$",8 laden und mit LIST ansehen. Mit

```
POKE 44,8:CLR
```

gelangen Sie dann wieder ins normale Programm zurück. Mit dieser Methode lassen sich auch andere Basic-Programme ohne Verlust des Hauptprogramms listen, allerdings nicht editieren, verändern oder starten.

C 64 - Computerlexikon

Absolutes Laden, das

Beim absoluten Laden einer Datei mit dem LOAD-Befehl muß hinter der Geräteadresse der Parameter »,1« (Beispiel: LOAD "NAME",8,1) angegeben werden. Diese Angabe bewirkt, daß das File in den Speicherbereich des C 64 gelangt, aus dem es auch gespeichert wurde, nicht unbedingt in den Basicspeicher (sog. »relatives Laden« ohne »,1«). Viele Maschinenprogramme sind nur in »ihrem« Speicherbereich und nicht im Basicspeicher lauffähig, das ist der Grund dafür, daß sie in der Regel absolut (»,8,1«) zu laden sind.

Absturz, der

Reagiert der Computer wegen eines Programmfehlers oder aufgrund anderer

Umstände nicht mehr auf den Anwender (Tastatur »tot«), hilft nur noch ein Kaltstart (»Reset«). Diese Situation wird »Absturz« genannt.

Akkumulator, der

Ein spezielles Register eines Prozessors, mit dem Rechenoperationen ausgeführt werden.

Algorithmus, der

Rechenregel, Verfahrensweise, Verfahren zur Lösung eines Problems, das nach endlich vielen Schritten abbricht und dabei entweder die Lösung des Problems präsentiert oder das Problem als unlösbar zurückweist.

Adresse, die

Die Adresse gibt an, an welcher Stelle im Speicher sich bestimmte Daten befinden (vergleichbar mit der Hausnummer in einer langen Straße). Die 65536 Speicherzellen des C 64 (65536 Häuser) sind durchnumeriert, wodurch ihnen Adressen zugeteilt werden, über die auf den Inhalt zugegriffen werden kann.

Alphanumerisches Zeichen, das

Darunter versteht man alle Zeichen, die ein Computer auf einer Ausgabeeinheit (Drucker, Bildschirm) darstellen kann; zumindest aber das Alphabet und die Dezimalzahlen.

ALU, die

»Arithmetical Logical Unit«; der für die Durchführung sämtlicher Berechnungen zuständige Teil des Prozessors.

Anweisung, die

Einzelner oder zusammengesetzter Ausdruck innerhalb eines Programms, der dem Computer mitteilt, was er zu tun hat.

ASCII

American Standard Code für Information Interchange. Ein Standardcode, der 128 Zeichen (Zahlen, Buchstaben, Satzzeichen, Symbole) ein aus sieben Bit bestehendes Muster zuweist. Dieser Standard erleichtert die Kommunikation verschiedener Programme, Computer und Peripheriegeräte.

Assembler, der

1. Eine maschinennahe Programmiersprache (Maschinensprache), die von Prozessor zu Prozessor unterschiedlich ist. Assemblerprogramme können nur mit »Ihrem« Prozessor laufen, im Gegensatz zu höheren Programmiersprachen, die unabhängig vom eingesetzten Mikroprozessortyp sind. Assemblerprogramme nutzen die Fähigkeiten des jeweiligen Computers optimal aus und belegen auch im Vergleich zu höheren Programmiersprachen wie etwa Basic weniger Speicherplatz und sind zudem sehr schnell.

2. Übersetzungsprogramm, welches ein im Assemblercode vorliegendes Programm (Quelltext) in den binären, also direkt ablauffähigen Maschinencode (Objektcode) umwandelt. Dabei werden die Assemblerbefehle (»Mnemonics«) auf syntaktische Richtigkeit überprüft.

Auflösung, die

Maximale Anzahl optisch voneinander unterscheidbarer Punkte, die auf einer Fläche dargestellt werden können. Angegeben wird die Auflösung in Pixel. Grafikcomputer höchster Auflösung können bis zu 4096 x 4096 Punkte auflösen, beim C 64 beträgt die Auflösung 320 x 200 Pixel, es stehen also 200 Zeilen zu je 320 Bildpunkten zur Verfügung.

BAM, die

Block Availability Map; Verzeichnis auf einer Diskette, welche Sektoren der Diskette von Daten belegt und welche frei sind.

Basic-Interpreter, der

Bei dem Basic-Interpreter handelt es sich um ein fest im C 64 eingebautes Maschinenprogramm mit der Aufgabe, die eingegebenen Basicbefehle zu entschlüsseln und auszuführen. Er belegt den Speicherbereich von 40960 bis 49151. Vgl. Betriebssystem

Betriebssystem, das

Das Betriebssystem (Kernal) ist ein fest im C 64 eingebautes Maschinenprogramm, das alle Ein- und Ausgaben steuert (Bildschirm, Tastatur, Diskette, Drucker und so weiter) und dafür sorgt, daß Anwendungsprogramme lauffähig und damit anwendbar sind. Es belegt den Speicherbereich von 57344 bis 65535. Bei anderen Computern muß das Betriebssystem (z.B. MS DOS) erst nach dem Start von Diskette geladen werden.

Bildschirmmaske, die

generelles Aussehen des Bildschirms bei einem bestimmten Programm, Formular

Binär

Zahlensystem mit der Basis 2, das nur die Ziffern 0 und 1 kennt. Da sich diese Zahlen leicht in einem Entweder-Oder-Schema darstellen lassen (Strom fließt - Strom fließt nicht), wurde das binäre Zahlensystem zum Charakteristikum digitaler Computersysteme.

Bit, das

»Binary Digit«; Kleinste Informationseinheit in einem binären Zahlensystem (eine Ziffer mit Wert »Null« (»gelöschtes Bit«) oder »Eins« (»gesetztes Bit«)).

Block, der

Sektor auf einer Diskette

Blockgrafik, die

Zeichensatzgrafik - der C 64 verfügt neben den ASCII-Zeichen über weitere Darstellungsformen: Die Grafiksymbole, die auf der Tastatur mit Hilfe der Tasten <SHIFT> und <CBM> erreicht werden können. Mit diesen zusätzlichen Zeichen lassen sich diverse nicht hochauflösende Grafiken bequem in Programme und Bildschirmmasken einbauen, beispielsweise Umrandungen etc.

Bus, der

Satz von acht oder 16 Leitungen in einem Computersystem, über das alle Bausteine verbunden sind. So werden beispielsweise über den Datenbus die Daten zwischen Speicher und Prozessor ausgetauscht.

Byte, das

Ein Byte ist eine aus acht Bit zusammengesetzte Informationseinheit, mit der sich jeder ganzzahlige Wert zwischen 0 und 255 darstellen (codieren) läßt. Jede Speicherzelle faßt ein Byte. Die nächstgrößere Verwaltungseinhait, das Kilobyte (KByte), besteht aus 1024 Bytes oder 8192 Bit. Der Speicher des C 64 faßt 64 Kilobyte, das entspricht 65536 Bytes oder 524288 Bit.

Chip, der

Rechteckiges Siliziumplättchen, das in einem Computer bestimmte elektrische Aufgaben erfüllt (Prozessor, Videochip, Speicherbaustein) und dabei meistens in eine integrierte Schaltung (IC) eingebaut ist.

CIA, die

»Complex Interface Adapter«, Baustein im C 64, der - in zweifacher Ausführung vorhanden - die elektronische Verbindung des Systems zu externen Komponenten wie Laufwerk, Drucker, Tastatur, Joystick, Userport, Datasette steuert. In der Zuständigkeit der CIAs liegt außerdem die Interrupt-Erzeugung und -Verwaltung. Die technische Bezeichnung für die CIA des C 64 lautet 6526.

Code, der

1. ein in Maschinensprache übersetztes Assemblerprogramm oder ein compiliertes Hochsprache-Programm

2. Vorschrift zum Ver-/Entschlüsseln von Daten

Compiler, der

Übersetzungsprogramm, das in einer Hochsprache (Beispiel Basic, Pascal) geschriebene Programme in Maschinensprache übersetzt und diese somit wesentlich schneller im Ablauf macht.

CPU, die

»central processing unit«, Teil des Prozessors

CRT, die

»cathode ray tube«, Kathodenstrahlröhre, gleichbedeutend mit »Bildschirm«.

Cursor, der
Das blinkende Viereck, das am Bildschirm die Schreibposition angibt.

Datei, die

engl. »file«; Sammlung von zusammengehörenden Informationen, die gemeinsam gespeichert sind. Ein auf Diskette gespeichertes Programm ist ebenso eine

Datei wie eine Adressenliste oder gar der gesamte Disketteninhalt. Man unterscheidet dabei hauptsächlich nach dem möglichen Zugriff in sequentielle Dateien (vgl. ein Tonband; um hier eine bestimmte Stelle zu erreichen, muß erst vorgespult werden) und in relative Dateien (vgl. eine Schallplatte, bei der der Tonarm in kürzester Zeit jede beliebige Stelle erreichen kann).

Datensatz, der

Teil einer Datei

Debugger, der

»Entwanzer«, Einrichtung zur Programm-Fehlersuche

Default, der

Vorgabewert, Vorschlag für einen Parameter. Beim LOAD-Befehl lautet beispielsweise der Default für die Geräteadresse 1 (Tape), daher versucht der C 64 von Datasette zu laden, wenn Sie keine Geräteadresse angeben.

Directory, das

Inhaltsverzeichnis zum Beispiel von einer Diskette; verzeichnet sind darin die gespeicherten Dateien. Beim C 64 wird es mit dem Befehl LOAD "$",8 geladen und dann mit LIST gezeigt.

Disassembler, der

Programm, das ein Maschinenprogramm in ein leicht lesbares Assemblerprogramm mit Mnemonics übersetzt, beispielsweise zur Wartung oder Fehlersuche. Oft Funktionsteil eines Monitors.

Diskette, die

Auch »Floppy Disc« genannt. Sie besteht aus einer flexiblen (»floppy«) Plastikscheibe mit einer magnetischen Beschichtung, auf die mit einem Tonkopf im Laufwerk magnetisch die Daten gespeichert werden. Der Vorteil von Diskette gegenüber Bändern besteht in der direkten Zugriffsmöglichkeit auf jeden Datensatz.

Diskettenmonitor, der

Programm, das es dem Anwender ermöglicht, Manipulationen von Daten direkt auf der Diskette vorzunehmen, Beispielsweise Dateien zu verändern oder Files oder auch die gesamte Diskette gegen Löschen oder Überschreiben zu schützen. Ebenso ist der Zugriff auf die BAM für den Programmierer nur mit einem Diskettenmonitor möglich.

DOS, das

engl. »Disc Operating System«; Betriebssystem einer Diskettenstation, das die mechanischen und elektronischen Funktionsabläufe im Drive steuert (Beispiel: Validieren einer Diskette).

Drive, das

engl. für »Diskettenlaufwerk«

Dummy, der

»Strohmann«, »Atrappe«; Platzhalter für einen Programmteil oder Daten, die erst später (z.B. bei der Initialisierung) in diesen Speicherbereich geschrieben werden, z.B. in der Testphase.

Editor, der

Programm, welches das Editieren von Programmen oder Texten oder Dateien erlaubt. Beispiel: Textverarbeitungsprogramm

Editieren, das

Anlegen oder Verändern einer Datei (z.B. Blöcke vertauschen, Zeichen einfügen oder löschen, kopieren von Textbereichen).

EPROM, das

»erasably programmable read only memory«, Speicherbaustein, der in einen Computer eingebaut nur gelesen, nicht aber beschrieben oder gelöscht werden kann. Zum Löschen des Bausteins wird spezielles UV-Licht verwendet, mit Hilfe eines »Eprom-Brenners« kann dann neuer Speicherinhalt in das Eprom geschrieben (»gebrannt«) werden.

File, das

siehe Datei

Flag, das

engl. »Fahne«, Signal; ein Flag kann nur zwei Zuständen annehmen: »wahr« oder »falsch«. Es signalisiert damit dem Programm bestimmte Betriebszustände.

Flipflop, das

Bezeichnung für eine elektronische Schaltung, die nur zwei Zustände besitzt. Der Übergang von einem Zustand in den anderen kann nur durch äußere Einflüsse erfolgen. Die Aufforderung dazu ergeht über mindestens einen Eingang, der Zustand des Flipflops kann über mindestens einen Ausgang erfaßt werden. Ein Flipflop ist ein Speicher für ein Bit, ein Beispiel für ein einfaches Flipflop ist ein Lichtschalter.

Floppy, die

siehe Diskette

Floppy-Speeder, der

Beschleuniger für ein Diskettenlaufwerk; die Diskettenlaufwerke für den C 64 zählen zu den langsamsten Laufwerken (sehr lange Lade- und Speicherzeiten), deshalb wurden im Laufe der Zeit einige Beschleuniger entwickelt, die die Geschwindigkeit des Datentransfers auf ein erträgliches Maß reduzieren. Manche dieser Speeder übertragen Dateien, für die das Original-Betriebssystem Minuten braucht, in nur wenigen Sekunden.

Garbage Collection, die

Bei der dynamischen Stringverwaltung entstehen bei fast jeder Textoperation kleine unbenutzte Speicherbruchstücke. Damit sie wieder genutzt werden können, müssen sie gesucht und dann zusammengefaßt werden. Diesen Vorgang, der beim C 64 durchaus einen scheinbaren Absturz von bis zu einer halben Stunden Dauer zur Folge haben kann, nennt man »Garbage Collection« (»Müllabfuhr«).

Geräteadresse, die

Parameter beispielsweise hinter einem LOAD- oder OPEN-Befehl, der angibt, auf welches Gerät sich der Befehl bezieht. Übliche Geräteadressen (Gerätenummern): 0 = Tastatur, 1 = Tape, 2 = RS232, 3 = Bildschirm, 4..7 = Drucker, 8..11 = Diskettenstation(en).

Hardcopy, die

Ausdruck von Texten oder Grafiken. Viele Computer (nicht der C 64) verfügen über eine spezielle Hardcopy-Funktion, mit der der aktuelle Bildschirminhalt auf dem Drucker ausgegeben werden kann. Beim C 64 gibt es entsprechende Programme (Hardcopy-Routinen), die geladen werden können und sich dann bei Bedarf per Tastendruck starten lassen.

Hardware, die

Überbegriff über alle Komponenten einer Computeranlage, die man anfassen kann, also die gesamte Elektronik, alle materiellen Teile (Gegenteil: Software)

Hexadezimal

Das hexadezimale Zahlensystem beschreibt die Darstellung von Zahlen zur Basis 16. Die Dezimalzahlen 0 bis 15 werden durch die Ziffern 0 bis 9 und die Buchstaben A bis F dargestellt. Dieses System wird oft zur platzsparenden Darstellung von Binärzahlen benutzt. Hexadezimalzahlen werden in der Regel durch ein vorangestelltes Dollarzeichen »$« markiert.

Highbyte, das

In einer Speicherzelle kann nur ein Wert bis 255 gespeichert werden. Sollen größere Zahlen gespeichert werden, muß man sie auf mehrere Speicherzellen aufteilen. Ein beliebtes Verfahren zum Speichern von Integerzahlen von 0 bis 65535 in zwei Zellen ist das High/Lowbyte-Verfahren. Dazu wird die Zahl nach folgenden Formeln zerlegt:

```
HIGHBYTE = INT (ZAHL/256)
LOWBYTE = ZAHL - HIGHBYTE * 256
```

Für den Fall, daß 0 <= ZAHL < 65536, liegen die Werte des High- und Lowbytes im Bereich von je 0 bis 255, sie sind also Speicherzellen-geeignet. Die beiden errechneten Bytes werden jetzt in zwei aufeinanderfolgenden Speicherzellen (erst Low-, dann Highbyte) gespeichert. Um aus den beiden die Zahl zu rekonstruieren, kommt folgende Formel zur Anwendung:

```
ZAHL = HIGHBYTE * 256 + LOWBYTE
```

Auf diese Weise speichert der C 64 Integervariablen. Sollen größere Zahlen als 65535 oder negative oder gebrochene Zahlen gespeichert werden, muß man die Werte mit geeigneten Formeln auf mehr als zwei Zellen aufteilen.

Hochsprache, die

Sprache mit starken Befehlssatz, zum Beispiel Ada, Algol, Apl, Basic, C, Cobol, Comal, Forth, Fortran, Lisp, Logo, Modula-2, Pascal, Smalltalk etc. Maschinensprache ist keine Hochsprache, da hier leistungsstarke und komplexe Befehle wie PRINT zur Ausgabe fehlen.

Initialisierung, die

Herstellung eines ganz bestimmten Startzustands eines Programms oder einer Prozedur. Dabei werden zum Beispiel die Variablen auf Null gesetzt.

Integer

ganzzahlig

Interface, das

Einrichtung, die zwei sonst nicht zueinander kompatible Komponenten miteinander verbindet. Beispiel: Druckerinterface zum Anschluß eines PC-Druckers an den C 64.

Interpreter, der

Der Interpreter durchläuft für jeden Befehl einer Programmiersprache eine festgelegte Befehlssequenz in Maschinensprache. Beispiel: Wird ein Basicprogramm ausgeführt und stößt der Interpreter dabei auf den PRINT-Befehl, so sorgt er dafür, daß seine entsprechende Bildschirmausgabe-Routine abgearbeitet wird.

Interrupt, der

Unterbrechung; definiertes Aussetzen eines Programmablaufs aufgrund eines von der Hardware kommenden Signals. Nach der Abarbeitung der »Interrupt-Routine« wird das unterbrochene Programm weitergeführt. Auf diese Art (eine CIA löst 60 Mal in der Sekunde über eine Leitung einen IRQ aus) wird beim C 64 das Cursorblinken, die Tastaturabfrage und die interne Uhr TI und TI$ gesteuert. Ein zu unterbrechendes Programm kann bestimmte Arten von Interrupts verhindern, »maskieren«. Interrupts, die sich ein- und ausschalten lassen, heißen »maskierbar«, Interrupts, die sich nicht verhindern lassen, heißen »nicht maskierbar« (beim C 64 löst die RESTORE-Taste einen NMI aus).

IRQ, der

Kurzform von »maskierbarer Interrupt«

Joystick, der

Eingabegerät, das nur fünf verschiedene Zuständen melden kann: Oben, unten, links, rechts, Feuer sowie Kombinationen daraus. Dieses primitive Gerät wird daher nur zur Steuerung von Spielen und wenigen Anwenderprogrammen verwendet.

Kaltstart, der

identisch mit Reset; Gegenteil: Warmstart

Kernal, das
»Kern«; siehe Betriebssystem

Kompatibilität, die

Eigenschaft von verschiedenen Hard- oder Softwarekomponenten, gegeneinander austauschbar zu sein oder gemeinsam zu einem System zusammengesetzt werden zu können. Speziell unterscheidet man Daten- und Anlagenkompatibilität. Läßt sich ein Drucker ohne weiteres an den C 64 anschließen, heißt er »kompatibel zum C 64«.

Label, das

Sprungmarke in einem Maschinenprogramm, Name einer Routine. In Basic gibt es keine Label, dafür Zeilennummern.

Lader, der

Hilfsprogramm, das ein Maschinenprogramm oder eine andere Datei in den Hauptspeicher bringt. Auf dem C 64 sind zwei Formen verbreitet: Der DATA-Lader, der in einer FOR..NEXT-Schleife mit READ und POKE DATA-Werte in den Speicher schreibt, und der Diskettenlader, der mit einem LOAD-Befehl ein Programm absolut in den Speicher lädt.

Leerstring, der

Leerstring bedeutet, daß sich in einer Textvariablen kein Inhalt befindet (A$ = "")

Lowbyte, das

siehe Highbyte

Maschinensprache, die

siehe Assembler

Maschinenroutine, die

Ein kleines in Maschinensprache geschriebenes Programm

Maus, die

Ein auf dem Tisch frei bewegliches, etwa faustgroßes Eingabegerät mit einer, zwei oder drei Tasten. Mit der Maus arbeitet es sich wesentlich komfortabler und einfacher als mit dem Joystick oder der Tastatur. Ein unten in der Maus angebrachter Gummiball überträgt durch Reibung die Bewegungen der Maus auf dem Tisch in die Maus, wo sie mit Sensoren abgetastet und über ein langes Kabel an den Computer gemeldet werden.

Mikroprozessor, der

siehe Prozessor

Mnemonic, das

griechisch für »Merkregel«, »Eselsbrücke«. Die Maschinensprache-Befehle wie

»LOAD ACCUMULATOR« werden durch Kurzbefehle (Mnemonics) wie »LDA« abgekürzt.

MOB, das

siehe Sprite

Monitor, der

1. Bildschirmeinheit eines Computersystems. Entspricht technisch in etwa einem Fernseher, aber ohne Tuner, dafür mit besserer, schärferer Bildwiedergabe.

2. Maschinensprache-Hilfsprogramm zum Darstellen und Ändern von Speicherinhalten. Siehe auch Diskettenmonitor.

Multitasking, das

Fähigkeit eines Systems, mehrere Aufgaben quasi gleichzeitig ausführen zu können. Vom Steuerungsprogramm werden dabei die verschiedenen »Jobs« (»Tasks«, Aufgaben) in hoher Geschwindigkeit umgeschaltet.

Nibble, das

ein halbes Byte, also vier Bit

NMI, der

nicht maskierbarer Interrupt. Wird beim C 64 beispielsweise durch die Restore-Taste ausgelöst. Die jetzt in jedem Fall aktivierte NMI-Routine prüft erst, ob gleichzeitig die RUN/STOP-Taste gedrückt wurde. Wenn nicht, geht es ohne Unterbrechung weiter, sonst wird ein Warmstart mit den bekannten Konsequenzen ausgeführt.

Nullbyte, das

Byte mit dem Wert Null; wird oft beispielsweise in Dateien oder Tabellen als Ende-Kennzeichen verwendet.

Page, die

»Seite« im Speicher (256 Byte)

Parameter, der

nähere Angabe zu einem Befehl

Peripherie, die

alle Geräte und Maschinen, die außer dem Grundgerät zum Computer gehören (Maus, Datasette, Joystick, Diskettenlaufwerk, Drucker, Bildschirm usw.)

Pixel, der

»picture element«, Bildpunkt, allerkleinste Einheit eines Grafikbildes.

Platine, die

Glasfaserverstärkte Kunststoffplatte, auf denen aufgedruckte oder aufgeätzte

Leiterbahnen anstelle von freien Leitungen die Verbindung zwischen den Baugruppen übernehmen. Die Bauteile werden durch Bohrungen in die Platine gesteckt und mit den Leiterbahnen verlötet.

Plotter, der

elektromechanisches Gerät zum Ausgeben von technischen Zeichnungen und Kurven. Über zwei Motoren wird dazu eine Art Kugelschreiber über das Papier bewegt.

Port, der

engl. »Hafen«; Schnittstelle zwischen Prozessor oder Computer und Außenwelt.

Prozessor, der

Zentrale Funktionseinheit innerhalb eines Computers, die alle anderen Komponenten steuert und überwacht. Die CPU (»central processing unit«) führt innerhalb des Prozessors die Maschinenprogramme aus, Berechnungen werden von der ALU durchgeführt. Der Prozessor spricht über den Bus den gesamten Speicher des Computers und ggf. andere Chips an. Die technische Bezeichnung des im C 64 eingebauten Prozessors lautet 6510.

RAM, das

»random access memory«, Speicher mit wahlfreiem Zugriff, der sowohl gelesen wie auch beschrieben werden kann. Hier speichert der Computer die eingegebenen oder geladen Programme und alle Daten. Nach dem Abschalten der Stromversorgung geht der Inhalt des RAMs verloren.

Real-Zahl, die

normale numerische Variable des C 64, die positive und negative Werte bis ca. 10 hoch 39 bei bis zu neun Stellen nach dem Komma annehmen darf. Zur Speicherung einer solchen Zahl werden genau fünf Byte benötigt.

Register, das

1. Speichereinheit eines Prozessors, die kurzzeitig Daten oder Zwischenergebnisse aufnehmen kann.

2. Speichereinheit eines Chips, die die Funktionsweise des Chips steuert. Etwa ist die bekannte Speicherzelle 53280 beim C 64 ein Register des Videobausteins VIC, das die Bildschirmrahmenfarbe steuert.

Rekursion, die

Definition eines Problems, einer Funktion oder eines Verfahrens (Algorithmus) durch sich selbst. Rekursive Darstellungen sind meist leichter verständlich und kürzer als andere Darstellungsformen. Ruft sich ein Programmteil mit GOSUB selbst auf, spricht man von rekursiver Programmierung.

Renew

Eine Renew-Routine ist ein praktisches Hilfsprogramm, mit dessen Hilfe in vielen Fällen nach einem versehentlich eingegebenen NEW-Befehl oder nach einem Kaltstart (Reset) das gelöschte Basicprogramm wiederhergestellt werden

kann.

Reset, der

Signal, Vorgang oder Befehl, um den Computer wieder in den Einschaltzustand zu versetzen. Wichtige Speicherzellen erhalten ihren Ausgangswert zurück, der Bildschirm, der Programmspeicher und alle Variablen werden gelöscht (siehe Renew). Von Basic aus kann beim C 64 der Reset mit dem Befehl SYS 64738 oder einem speziellen Reset-Taster (vgl. Bauanleitung) ausgelöst werden.

ROM, das

»read only memory«, Speicher, der nur gelesen werden kann und dessen Inhalt nach dem Ausschalten nicht verlorengeht (»nicht flüchtig«). Im C 64 befinden sich drei ROMs (in neueren Versionen nur noch zwei), in denen der Bildschirmzeichensatz, der Basic-Interpreter und das Kernal (Betriebssystem) gespeichert sind.

Routine, die

Unterprogramm, Programmteil

Schleife, die

Programmteil, der so oft immer wieder durchlaufen wird, bis eine bestimmte Abbruchbedingung erfüllt wird.

Schnittstelle, die

Meist steckbare elektrische Verbindung zwischen zwei Hardware-Komponenten, Anschluß

Scrollen, das

auch »Scrolling«; gemeint ist das Hochrollen des Bildschirm beim Listen eines Programms oder einer Tabelle, die länger als eine Bildschirmseite ist.

Seite, die

Eine Seite (»page«) im Speicher umfaßt 256 Bytes. Siehe auch zeropage.

Sektor, der

auch »Block«; nach dem Track nächstkleinere Verwaltungseinheit für Datenspeicherung auf Disketten. Jeder Track ist je nach Tracknummer in 17 bis 21 Sektoren unterteilt. Ein Sektor kann 254 Daten- und zwei Verwaltungsbytes speichern. Die beiden Verwaltungsbytes geben die Nummer des Tracks und Sektors an, wo dieser Block fortgesetzt wird.

Sensor, der

Aufnehmer, Fühler

SID, der

»Sound Interface Device«, Baustein im C 64, der den Ton erzeugt. Die technische Bezeichnung dieses ICs lautet 6581.

Software, die

nicht materieller Teil eines Computers, also die Daten und die Programme (Gegenteil: Hardware)

Speicher, der

Man unterscheidet den Haupt- oder Arbeitsspeicher (meist) im Gehäuse des Computers und die anzuschließende Massenspeichereinheit (Diskette oder Magnetband). Der Hauptspeicher funktioniert wie ein großes Regal mit numerierten Fächern (Adressen). So wie der Postbote bei der Zustellung eines Briefes die Adresse mit Hausnummer wissen muß, kennt der Computer beim Speichern und Lesen von Daten die Adressen der Speicherzellen. In einer Speicherzelle kann ein Byte (entspricht einem Zeichen) abgelegt werden.

Sprite, das

Vom Programmierer frei definierbares grafisches Objekt. Auf dem C 64 ist ein Sprite 21 x 24 Punkte groß. Sprites können dort sowohl grafisch wie auch schwarz/weiß dargestellt werden. Der Vorteil dieser Gebilde besteht darin, daß die unabhängig vom sonstigen Bildschirminhalt als eigenständige Fläche bewegt werden können. Ohne programmtechnische Tricks können bis zu acht dieser »MOBs« (movable Object) dargestellt werden.

Spur, die
andere Bezeichnung für »Track«

Stack, der
Siehe Stapelspeicher

Stapelspeicher, der

engl. »Stack«; Organisationsform eines Speichers, der nach dem LIFO-Verfahren arbeitet: »Last in, first out«. Die Elemente, die als letztes im Stack gespeichert wurden, kommen als erste wieder heraus. Stellen Sie sich einen Stapel Papier vor: Sie legen ein neues Blatt oben auf. Soll jetzt ein Element aus dem Stapel genommen werden, muß es sich um das oberste handeln, also das, was eben erst draufgelegt wurde. Beim C 64 arbeitet beispielsweise der GOSUB-Befehl mit einem Stapelspeicher, in dem die Aufrufadressen abgelegt werden. Der RETURN-Befehl holt sich vom Stapel die Basiczeile, in der der letzte GOSUB-Befehl stand. Auf diese Weise lassen sich GOSUB-Befehle verschachteln. Andere Beispiele für Stacks: FOR..NEXT-Schleifen in Basic oder Unterprogramme in Maschinensprache mit JSR..RTS.

Startadresse, die

Adresse des ersten auszuführenden Befehles in einem Programm

String, der

Text, Zeichenkette (engl. »Kette«)

Stringvariable, die

In einer Stringvariablen lassen sich Zeichen speichern. Im Gegensatz zu den normalen numerischen Variablen kann man Stringvariablen weder addieren noch

multiplizieren. Wichtig ist, daß Zeichen hier innerhalb von Hochkommas einzugeben sind. Beispiel: A$ = "PETER"

Systemtakt, der

siehe Takt

Swoboda, der

Informatik-Professor vom Lehrstuhl für Datenverarbeitung an der TU München

Takt, der

Der (System-)takt steuert innerhalb des Computers alle zeitlichen Abläufe und dient so zur Synchronisation. Vereinfacht ausgedrückt wird pro Takt ein Maschinenbefehl ausgeführt. Beim C 64 beträgt der Takt knapp 1 Million pro Sekunde, es können also fast 1 Million Befehle pro Sekunden (in Assembler!) ausgeführt werden. Der Takt wird auch im VIC zur Bilderzeugung verarbeitet.

temporär
vorübergehend, zeitweilig

Track, der

Spur auf einer Diskette. Jede Diskette des C 64-Laufwerks ist in 41 konzentrische Kreise unterteilt, die zur Datenspeicherung verwendet werden. Der Tonkopf kann mit einem eigenen Motor in 41 x 2 = 82 Schritten auf jede der 41 Spuren und dazwischen (»Halftracks«) positioniert werden und so mit relativ hoher Zugriffszeit die gespeicherten Daten lesen oder verändern. Vom System werden nur die Spuren 1 bis 35 verwendet. Zu Kopierschutzzwecken oder um mehr Daten speichern zu können werden von einigen kommerziellen Programmen auch die physikalisch durchaus vorhandenen Spuren 36 bis 41 verwendet. Der C 64 kann dies mit Programmen wie beispielsweise dem DMS auch.

Update, das

Neue, ggf. verbesserte Version eines kommerziellen Programms

Userport, der

Schnittstelle des C 64, die sich hinten links befindet. Der Userport enthält unter anderem je zwei Interrupt-Leitungen und zwei serielle Ports, sowie vor allem einen neun Bit breiten Datenport, dessen neun Leitungen unabhängig voneinander jeweils als Aus- oder Eingänge arbeiten können. Der Userport wird gern von Bastlern genutzt, um eigene Erweiterungen wie Eprom-Brenner, Modem, Akkustikkoppler, Echtzeituhr, Eisenbahn-Steuerung und dergleichen an den C 64 anzuschließen. Bei Fehlbedienung des Userport (zum Beispiel Berührung oder Überspannung) wird in den meisten Fällen eine CIA zerstört.

Variable, die

symbolischer Name, an den ein Wert oder eine Zeichenkette gebunden ist. Indem sich der Programmierer auf den Namen einer Variablen bezieht, bezieht er sich auf den in der Variablen enthaltenen Wert. Es gibt verschiedene Variablenarten, je nachdem, was darin gespeichert werden kann. So ist eine Integervariable nur zur Aufnahme ganzer Zahlen in einem bestimmten Bereich geeignet.

VIC, der

»Video Interface Controller«, Baustein im C 64, der das Grafikbild erzeugt und direkt an den Monitor leitet. Die technische Bezeichnung dieses ICs lautet 6569.

Warmstart, der

Übergang des Computers in einen definierten Einschaltzustand, wobei allerdings Programme und Daten nur unterbrochen, nicht aber gelöscht werden (im Gegensatz zum Kaltstart). Beim C 64 lösen Sie mit der bekannten Tastenkombination <RUN STOP/RESTORE> einen Warmstart aus.

Zeichensatz, der

Datei, die das Aussehen der Zeichen auf dem Bildschirm oder auch auf dem Drucker verfügt

Zeropage, die

Speicherbereich beim C 64 von Adresse 0 bis 255 (manchmal wird auch der Bereich von 0 bis 1023 als Zeropage bezeichnet). Hier befinden sich wichtige Betriebsdaten zur Steuerung des Basic-Ablaufs und des Basic-Editors. Durch Manipulationen in diesen Speicherzellen läßt sich Einfluß auf fast alle internen Vorgänge nehmen.

****

Directory-Content of the two Discs
with the book "222 Tips, Tricks, Tools fuer den C 64"

Disc 1 of 2:

```
0 "64'ER BUCH DISK1" IPV92
32   "LIES MICH!"       PRG
9    "MESS.FONT"        PRG
45   "MESS.MUSIC"       PRG
0    "----------------" DEL
0    "-  KAPITEL 2   -" DEL
0    "----------------" DEL
0    "-  ASSEMBLER   -" DEL
0    "----------------" DEL
1    "ASS BEISPIEL 1"   PRG
1    "ASS BEISPIEL 2"   PRG
1    "ASS BEISPIEL 3"   PRG
1    "ASS BEISPIEL 4"   PRG
1    "ASS BEISPIEL 5"   PRG
1    "ASS BEISPIEL 6"   PRG
1    "ASS BEISPIEL 7"   PRG
0    "----------------" DEL
2    "MT 49152"         PRG
2    "MT.DEMO"          PRG
2    "MT.DEMO.MC"       PRG
2    "MT.DEMO.SP"       PRG
18   "OP.MULTITASK (K)" PRG
0    "----------------" DEL
33   "NSS KERNAL 2764"  PRG
```

```
65   "NSS KERNAL 27128" PRG
0    "----------------" DEL
3    "EISBERG"          PRG
20   "OP.EISBERG"       PRG
0    "----------------" DEL
3    "UNCRASH"          PRG
26   "OP.UNCRASH (K)"   PRG
0    "----------------" DEL
5    "QUICKIE-GENERATO" PRG
2    "QUICK-GEN $7000"  PRG
4    "QUICKIE DEMO"     PRG
7    "OP.QUICKIE"       PRG
0    "----------------" DEL
33   "NEUES DOS $E000"  PRG
0    "----------------" DEL
0    "-  KAPITEL 3   -" DEL
0    "----------------" DEL
0    "-    GRAFIK    -" DEL
0    "----------------" DEL
14   "SPRITELIST V2"    PRG
1    "SP.AXT"           PRG
1    "SP.BRATEN"        PRG
1    "SP.DISK"          PRG
1    "SP.MUELLEIMER"    PRG
1    "SP.SNOOPY"        PRG
0    "----------------" DEL
12   "CHARTRANSPOSER"   PRG
9    "ZS.LINKISCH"      PRG
9    "ZS.MINI"          PRG
9    "ZS.OCR"           PRG
9    "ZS.RAMS"          PRG
9    "ZS.RUSSIAN"       PRG
0    "----------------" DEL
1    "SOFTSCROLL"       PRG
3    "OP.SOFTSCROLL"    PRG
0    "----------------" DEL
2    "DOUBLE PRINT"     PRG
5    "OP.DOUBLE PRINT"  PRG
0    "----------------" DEL
4    "LETTER V2 $C000"  PRG
4    "LETTER V2 DEMO"   PRG
16   "OP.LETTER"        PRG
0    "----------------" DEL
2    "USING"            PRG
8    "USING-DEMO"       PRG
9    "OP.USING"         PRG
0    "----------------" DEL
0    "-  KAPITEL 5   -" DEL
0    "----------------" DEL
0    "-  SONSTIGES   -" DEL
0    "----------------" DEL
13   "DOC 64 $8000"     PRG
52   "OP.DOC 64 2.2"    PRG
0    "----------------" DEL
2    "KALENDER PRG. 1"  PRG
2    "KALENDER PRG. 2"  PRG
2    "KALENDER PRG. 3"  PRG
2    "KALENDER PRG. 4"  PRG
```

```
3    "KALENDER PRG. 5"  PRG
2    "KALENDER PRG. 6"  PRG
1    "KALENDER SR 1000" PRG
2    "KALENDER SR 2000" PRG
0    "----------------" DEL
2    "CONDENSED"        PRG
9    "CONDENSED.DEMO"   PRG
26   "OP.CONDENSED"     PRG
0    "----------------" DEL
12   "MESSAGE-MAKER"    PRG
3    "MESS.KOPF"        PRG
45   "OP.MESSAGE-MAKER" PRG
13   "OP.MESS-KOPF"     PRG
0    "----------------" DEL
0    "-     ENDE     -" DEL
0    "-    DISK 1    -" DEL
0    "----------------" DEL
23 BLOCKS FREE.
```

Disc 2 of 2:

```
0 "64'ER BUCH DISK2" IPV92
0    "----------------" DEL
0    "-  KAPITEL 1   -" DEL
0    "----------------" DEL
0    "-    BASIC     -" DEL
0    "----------------" DEL
13   "BKS 5.0 (49152)"  PRG
22   "BKS.WHAT 5.0"     PRG
0    "----------------" DEL
1    "HELP !!"          PRG
18   "BASIC.MSG"        PRG
32   "OP.HELP"          PRG
0    "----------------" DEL
4    "DATA AID"         PRG
3    "DATA AID DEMO"    PRG
13   "OP.DATA AID"      PRG
0    "----------------" DEL
3    "TI$"              PRG
29   "OP.TI$"           PRG
0    "----------------" DEL
7    "REM-KILLER++"     PRG
27   "OP.REM-KILLER"    PRG
0    "----------------" DEL
2    "64 KEYS (LADER)"  PRG
9    "OP.64 KEYS (K)"   PRG
0    "----------------" DEL
5    "ARRAY OF BYTE"    PRG
0    "----------------" DEL
1    "INFORM 49152"     PRG
2    "INFORM DEMO"      PRG
13   "OP.INFORM"        PRG
0    "----------------" DEL
3    "GENIE-SUCH"       PRG
7    "GENIE-DEMO"       PRG
14   "OP.GENIE-SUCH"    PRG
0    "----------------" DEL
```

```
0    "-  KAPITEL 4   -" DEL
0    "----------------" DEL
0    "-    FLOPPY    -" DEL
0    "----------------" DEL
15   "REL-DEMOPROGRAMM" PRG
0    "----------------" DEL
14   "ARC 1.5"          PRG
0    "----------------" DEL
4    "BUMPMASTER"       PRG
0    "----------------" DEL
1    "DON'T REPLACE!"   PRG
5    "OP.DON'T REPLACE" PRG
0    "----------------" DEL
15   "TSS 2.1"          PRG
55   "OP.TSS 2.1"       PRG
0    "----------------" DEL
5    "VERIFY 2 FILES"   PRG
20   "OP.VERIFY 2 FILE" PRG
0    "----------------" DEL
5    "UNSCRATCH"        PRG
15   "OP.UNSCRATCH"     PRG
0    "----------------" DEL
7    "TOP SECRET"       PRG
26   "OP.TOP SECRET"    PRG
0    "----------------" DEL
16   "DISK SPY"         PRG
0    "----------------" DEL
21   "DMS"              PRG
0    "----------------" DEL
14   "TESTER 1541"      PRG
0    "----------------" DEL
0    "-     ENDE     -" DEL
0    "-    DISK 2    -" DEL
0    "----------------" DEL
198 BLOCKS FREE.
```